ALGORITHMS FOR BUDGETED AUCTIONS AND MULTI-AGENT COVERING PROBLEMS

A Thesis
Presented to
The Academic Faculty

by

Gagan Goel

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Algorithms, Combinatorics, and Optimization

Georgia Institute of Technology
August 2009

ALGORITHMS FOR BUDGETED AUCTIONS AND MULTI-AGENT COVERING PROBLEMS

Approved by:

Professor Vijay Vazirani, Advisor
College of Computing
Georgia Institute of Technology

Professor Robin Thomas
School of Mathematics
Georgia Institute of Technology

Professor Prasad Tetali
School of Mathematics
Georgia Institute of Technology

Professor Adam Kalai
College of Computing
Georgia Institute of Technology

Professor Milena Mihail
College of Computing
Georgia Institute of Technology

Date Approved: 15 June 2009

ACKNOWLEDGEMENTS

I would like to thank my advisor, Vijay Vazirani, for all his guidance, and from whom I have learnt the most. Especially during the last two years, he has played the role of a great mentor and a friend, always giving the right advice at the right moment. His knowledge and wisdom about research and life is among the best, and I feel lucky to have acquired at least a few bits of that, both consciously and subconsciously.

Thanks to Milena Mihail for her support especially when I needed it the most, the initial phases when I would think of quitting. Thanks to Aranyak Mehta for being a great collaborator, friend, and mentor during my internship at Google Inc. My initial work with him, in many ways, defined my research interests. Thanks to my co-authors and friends Deeparnab Chakrabarty, Lei Wang, Chinmay Karande, and Pushkar Tripathi for all the great moments we shared working together and writing papers. Thanks to Adam O'Neill, Anuj Madan, Atish Das Sarma, David Cash, Deepak Jahagirdar, Himani Goel, Luis Rademacher, Munish Gupta, Rahul Ghosh, Rishi Saket, Subruk, and many other friends with whom I have shared some of the best moments in Atlanta. Thanks to Adam O'Neill for all the philosophical breaks. Thanks also to the rest of the theory and ACO group members for creating a great research environment.

Finally, thanks to my parents and family members for always being there, even though I could not never explain to them what exactly I was working on all these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In this thesis, we do an algorithmic study of optimization problems in budgeted auctions, and some well known covering problems in the multi-agent setting. We give new results for the design of approximation algorithms, online algorithms and hardness of approximation for these problems. Along the way we give new insights for many other related problems.

Budgeted Auctions. We study the following allocation problem which arises in budgeted auctions: Given a set of m indivisible items and n agents; agent i is willing to pay b_{ij} for item j and has an overall budget of B_i (i.e. the maximum total amount agent i is willing to pay). The goal is to allocate items to the agents so as to maximize the total revenue obtained.

We give two approximation algorithms with 3/4-approximation factor, improving upon the previous best known factor of $\simeq 0.632$. We use linear programming based techniques, and our approximation ratio is optimal in the sense that it matches the integrality gap of the linear program used by us and by the other authors [24, 2, 3]. We also show hardness of approximation factor of 15/16 for the above problem, and use our techniques to improve the hardness results for other related allocation problems.

We also study the above allocation problem in an online setting. Online version of the problem has motivation in Ad Auctions which are run by search engines such as Google, Microsoft Live, Yahoo!. Previously, the online version was mostly studied in the worst-case setting. But in applications such as Ad Auctions it is unlikely to see a worst-case sequence of items as input. Motivated by this, we study the online version

of the problem in which the items arrive in a *random* order i.e. the set of items are adversarially chosen but the order in which they arrive is random. Our main result is an analysis to show that a natural greedy algorithm has a competitive ratio of $1-1/e$ in this model, which we also show to be tight.

Lastly, we define a new bidding language for budgeted auctions- Decreasing valuation bids. We make a case for why this language is better both for the sellers and the buyers.

Multi-agent Covering Problems. To motivate this class of problems, consider the network design problem of constructing a spanning tree of a graph, assuming that there are many agents willing to construct different parts of the tree. The cost of each agent for constructing a particular set of edges could be a complicated function. For instance, some agents might provide discounts depending on how many edges they construct. The algorithmic question that one would be interested in is: Can one find a spanning tree of minimum cost in polynomial time in these complex settings? Note that such an algorithm will have to find a spanning tree, and partition its edges among the agents.

Above are the type of questions that we are trying to answer for various combinatorial problems. We look at the case when the agents' cost functions are submodular. A function $f : 2^X \rightarrow R$ is said to be submodular if, for all $S, T \subseteq X$, $f(S \cap T) + f(S \cup T) \leq f(S) + f(T)$. Submodular functions form a rich class and capture the natural properties of economies of scale or the law of diminishing returns. Based on these considerations, we define the following general class of problems - We are given a set of elements X ($|X| = n$) and a collection $C \subseteq 2^X$. Here the collection C is given via some combinatorial structure like a matroid or a graph property (for instance the set of all spanning trees in a graph G with edge set X). We are also

given k agents, where each agent i specifies a non-decreasing submodular cost function $f_i : 2^X \rightarrow R^+$. The goal is to find a set $S \in C$ and a partition $S_1, ..., S_k$ of S such that $\sum_i f_i(S_i)$ is minimized. Note that one can study various covering type problems in this setting by suitably defining the collection set C.

We study the following fundamental problems in this setting- *Vertex Cover, Spanning Tree, Perfect Matching, Reverse Auctions*. We look at both the single agent and the multi-agent case, and study the approximability of each of these problems. The approximation ratio(upper bound) and the hardness of approximation(lower bound) for each of these problems is presented in the table below.

Table 1: Results

	Single-agent		Multi-agent	
	Lower bound	Upper bound	Lower bound	Upper bound
Reverse Auction	1	1	$\Omega(\log n)$	$O(min(k, \log n))$
Vertex Cover	$2 - \epsilon$	2	$\Omega(\log n)$	$O(\log n)$
Perfect Matching	$\Omega(n)$	$O(n)$	$\Omega(n)$	$O(n)$
Spanning Tree	$\Omega(n)$	$O(n)$	$\Omega(n)$	$O(n)$

CHAPTER I

INTRODUCTION

Algorithm Design is the field of designing step-by-step procedures to achieve certain outcome. One typical and important characteristic requirement is that the algorithm be efficient i.e. it takes *small* number of steps. Formalism of the notion of efficient algorithms has made a significant impact in many areas, besides Computer Science, such as Operations Research, Economics, Game Theory, etc. The proof for existence and non-existence of such algorithms typically involves deeper understanding of the problem at hand, thus giving valuable insights for both theory and practice.

In this thesis, we do an algorithmic study of optimization problems in budgeted auctions, and some well known covering problems in the multi-agent setting. We design approximation algorithms, online algorithms and show hardness of approximation for these problems. Along the way we give new insights for many other related problems.

1.1 Budgeted Auctions

Budgeted Auction is an auction setting where we are given a set of m indivisible items and n agents who are interested in the items; agent i bids b_{ij} on item j (i.e. the maximum amount agent i is willing to pay for item j) and has an overall budget of B_i (i.e. the maximum total amount "he" is willing to pay).

Examples of such auctions include those used for the privatization of public assets in western Europe (see, for example [6]), or those for the distribution of radio spectra in the US, where the magnitude of the transactions involved put financial or liquidity constraints on bidders. With the growth of the Internet, budget-constrained auctions have become increasingly relevant. Firstly, e-auctions held on the web (on e-Bay,

for instance) cater to the long-tail of users who are inherently budget-constrained. Secondly, sponsored search auctions hosted by search engines (Google, Yahoo!, MSN and the like) where advertisers bid on keywords include budget specification as a feature. Google also uses budgeted auctions to sell advertisement spaces on more traditional media such as radio and television [1].

A typical goal of the seller is to allocate items to the agents so as to maximize the total revenue. In this work, we focus our attention on the computational complexity of this revenue maximizing problem. Note that we do not consider the strategic nature of the agents (In general, the agents are strategic players who might lie on the bids or the budgets to maximize their own gains). We will denote this non-strategic optimization problem of budgeted auctions by **budgeted allocation**.

We study budgeted allocation problem in both the offline and the online setting. In the online setting, we will assume that the bids of advertisers are much smaller as compared to the budgets. This is a common (and natural) assumption made in the context of sponsored search auctions, which motivated the study of online budgeted allocation problem.

1.2 Multi-agent Submodular Covering Problems

A multitude of fundamental computational problems with real-world applications can be cast in the following framework: We are given a set X of elements, a collection C of subsets of X (i.e. $C \subseteq 2^X$) and a cost function f over the subsets of X. The collection C is typically specified via a combinatorial structure like a matroid or a graph property (for instance, the set of all spanning trees in a graph). The objective is to select a set $S \in C$ that minimizes $f(S)$.

A major focus in theoretical computer science has been on linear cost functions.

[1] see for instance `http://www.google.com/adwords/audioads/` and `http://www.google.com/adwords/tvads`

The study of combinatorial problems with linear cost functions has led to great developments in the theory of exact and approximation algorithms. However, linear cost functions do not always model the complex dependencies of the costs in a real-world setting. Often, they only serve as an approximation to the original functions. As a result, even though we might have a good algorithm for solving some linear optimization problem, the lack of accurate input might lead to sub-optimality of the solution.

Another feature that arises in a real-world setting is the presence of multiple agents, where each agent has his own cost function. Thus, in the optimal solution, each agent might build only a part of the required combinatorial structure. For example, the Internet is a complex multi-agent system where each service provider owns only a part of the network. For linear cost functions, it is easy to see that having multiple agents does not change the complexity of the original problem. However, this is not the case for more general cost functions.

Motivated by these considerations, we define the following class of *combinatorial problems with multi-agent submodular cost function* (MSCP) problems - We are given a set of elements X ($|X| = n$) and a collection $C \subseteq 2^X$. We are also given k agents, where each agent i specifies a normalized (meaning $f_i(\phi) = 0$) non-decreasing submodular cost function $f_i : 2^X \rightarrow R^+$. The goal is to find a set $S \in C$ and a partition $S_1, ..., S_k$ of S such that $\sum_i f_i(S_i)$ is minimized.

We study the following fundamental problems in this setting- *Vertex Cover, Spanning Tree, Perfect Matching, Reverse Auctions*. We look at both the single agent and the multi-agent case, and study the approximability of each of these problems. We assume that the submodular functions are given via a *value* oracle, i.e. given a set S, the oracle returns the value $f(S)$.

1.3 Contributions, Credits, and Organization of the thesis

1.3.1 Budgeted Auctions

As mentioned earlier we study the non-game theoretic maximum budgeted allocation (MBA) problem which arises in budgeted auctions.

In Chapter 2, we look at the offline version of the problem, i.e. when all the items are known. This is joint work with Deeparnab Chakrabarty [13] (Also appeared in the Ph.D. thesis of Chakrabarty [12]). We give two approximation algorithms for maximum budgeted allocation. The first, based on iterative LP rounding, attains a factor of 3/4. The second algorithm, based on the primal-dual schema, is faster and attains a factor of $(3/4)(1-\epsilon)$, for any $\epsilon > 0$. The running time of the latter algorithm is $\tilde{O}(\frac{nm}{\epsilon})$, and is thus almost linear for constant ϵ and dense instances. Our algorithms are optimal for the LP that we consider, as there is an example with an integrality gap of 3/4. Our algorithms can be extended to give a $1 - \beta/4$ factor approximation algorithm, when the ratio of bid to budget is at most β, where $0 < \beta \leq 1$ (We call this problem β-MBA).

We also show that it is NP hard to approximate MBA to a factor better than 15/16. Our hardness reductions extend to give a $(1 - \beta/16)$ hardness for β-MBA as well. Interestingly, our reductions can be used to obtain better inapproximability results for other problems: Submodular Welfare Maximization (15/16 hardness even with demand queries), Generalized Assignment Problem (10/11 hardness) and Maximum Spanning Star Forest(10/11 and 13/14 for the edge and node weighted versions).

Previous to our work, MBA was noted to be NP-hard and this observation was made concurrently by many authors ([24, 2, 45]). The first approximation algorithm for the problem was given by Garg, Kumar and Pandit[24] who gave a $2/(1+\sqrt{5})$ ($\simeq$ 0.618) factor approximation. Andelman and Mansour[2] improved the factor to $(1 - 1/e)$ ($\simeq$ 0.632). For the special case when budgets of all bidders were equal, [2] improved the factor to 0.717. Recently, and independent of our work, Azar et.al. [3]

obtained a 2/3-factor algorithm for the general MBA problem (Arvind Srinivasan [59] improved the algorithm to get a factor 3/4, same as ours). They also considered a *uniform* version of the problem where for every item j, the bid of any agent is either b_j (independent of the agent) or 0. They gave a $1/\sqrt{2}$ ($\simeq 0.707$) factor for the same.

Because of the applications in sponsored search auctions, MBA (or rather β-MBA with $\beta \to 0$ which is a standard assumption in such auctions) is also considered in an online setting, i.e. when items arrive one by one, and they have to be allocated as soon as they arrive. Mehta et.al.[48] and later, Buchbinder et.al.[9] gave $(1 - 1/e)$-competitive algorithms for the worst case scenario when each arriving item is chosen adversarially. Simple Greedy, which assigns the item to the highest bidder with remaining budget, has a competitive ratio of 1/2. In spite of the elegance and optimal competitive ratio of this algorithm, there remains some dissatisfaction with the use of worst case analysis in that we are unlikely to see a worst-case sequence of items as input.

Motivated by above considerations, in Chapter 3, we study the online budgeted allocation problem in which the items arrive in a *random* order, i.e. the set of items are adversarially chosen but the order in which they arrive is random. This is joint work with Aranyak Mehta [26].

Our main result is an analysis to show that Greedy has competitive ratio $1 - 1/e$ in the random permutation input model. We also show that our analysis is tight by providing examples in the two models for which Greedy has ratio exactly $1 - 1/e$. We also extend these results for the i.i.d. model in which there is an unknown distribution on items, and the next item in the sequence is picked independently from this distribution. Along the way, we provide a first correct proof for the RANKING algorithm of [37] for the online bipartite matching problem.

In Chapter 4, we study a new bidding model for the sponsored search auctions – *decreasing valuation bids*. This provides a richer language than the current model for

advertisers to convey their preferences. This is also a joint work with Aranyak Mehta [25].

Better expressivity is clearly better for the bidder (as long as the language remains simple enough to understand). Intuitively, it is clear that the bidders will now be able to bid with more control and therefore face less risk, and will bid more aggressively. This is clearly better for the search engine, in terms of the optimal profit (OPT) derivable from the bidders. But what if the bidding language introduces computationally difficult problems for the search engine? Then it will not be able to efficiently extract a good portion of the OPT as profit. We show that our models do not introduce such computational difficulties, by describing optimal algorithms whose competitive ratio is $1 - 1/e$, as good as that of the optimal algorithm [48] in the standard model.

We also show that our bidding language is at the correct trade-off point between expressivity, simplicity and computational efficiency. Simple generalizations of our bidding model (by adding more expressivity) result in computational problems for which no algorithms can perform better than a factor of $1/2$, and for which the natural Greedy algorithm has an arbitrarily bad factor.

1.3.2 Multi-agent Submodular Covering Problem

The following contribution is described in chapter 5. In a joint work with Chinmay Karande, Pushkar Tripathi, and Lei Wang, we give an approximation algorithm and a matching information theoretic lower bound for each of the subclass of problems that we mentioned earlier. We present the approximation ratio (both upper and lower bound) for each of these problems in the table below. Upper bound of $\log n$ was previously known for Reverse Auctions[30].

Quite surprisingly, Perfect Matching and Spanning Tree, which are polytime solvable in the linear cost setting, become extremely hard when the cost functions are submodular. On the other hand, Vertex Cover retains its factor 2 approximation

	Single-agent		Multi-agent	
	Lower bound	Upper bound	Lower bound	Upper bound
Reverse Auction	1	1	$\Omega(\log n)$	$O(min(k, \log n))$
Vertex Cover	$2-\epsilon$	2	$\Omega(\log n)$	$O(\log n)$
Perfect Matching	$\Omega(n)$	$O(n)$	$\Omega(n)$	$O(n)$
Spanning Tree	$\Omega(n)$	$O(n)$	$\Omega(n)$	$O(n)$

ratio. Note that factor $2-\epsilon$ hardness of approximation for Vertex Cover is a long standing open question in the classical linear cost setting. Our results implies that, if the cost function over the set of vertices is submodular, which is the first natural generalization of linear costs, then the optimal approximation factor is indeed 2.

Our algorithms are either based on greedy methods, or rounding of Configurational LPs; a key fact we use is that one can solve submodular function minimization in polynomial time.

CHAPTER II

BUDGETED ALLOCATION: OFFLINE CASE

In this chapter, we study the approximability of maximum budgeted allocation problem and improve upon the best known approximation and hardness of approximation factors. Moreover, we use our hardness reductions to get better hardness results for other allocation problems like submodular welfare maximization(SWM), generalized assignment problem (GAP) and maximum spanning star-forest (MSSF).

We start with the formal problem definition of maximum budgeted allocation(MBA).

Definition 1 *Let Q and A be a set of m indivisible items and n agents respectively, with agent i willing to pay b_{ij} for item j. Each agent i has a budget constraint B_i and on receiving a set $S \subseteq Q$ of items, pays* $\min(B_i, \sum_{j \in S} b_{ij})$. *An* allocation *$\Gamma : A \to 2^Q$ is the partitioning the sets of items Q into disjoint sets $\Gamma(1), \cdots, \Gamma(n)$. The* maximum budgeted allocation *problem, or simply MBA, is to find the allocation which maximizes the total revenue, that is,* $\sum_{i \in A} \min(B_i, \sum_{j \in \Gamma(i)} b_{ij})$.

Note that we can assume without loss of generality that $b_{ij} \leq B_i,\ \forall i \in A, j \in Q$. This is because if bids are larger than budget, decreasing it to the budget does not change the value of any allocation. Sometimes, motivated by the application, one can add the constraint that $b_{ij} \leq \beta \cdot B_i$ for all $i \in A$ and $j \in Q$, for some $\beta \leq 1$. We call such an instance β-MBA.

As mentioned earlier, the problem naturally arises as a revenue maximization problem for the auctioneer in an auction with budgeted agents. Examples of such auctions (see, for example [6]) include those used for the privatization of public assets in western Europe, or those for the distribution of radio spectra in the US, where

the magnitude of the transactions involved put financial or liquidity constraints on bidders. With the growth of the Internet, budget-constrained auctions have gained increasing relevance. Firstly, e-auctions held on the web (on e-Bay, for instance) cater to the long-tail of users who are inherently budget-constrained. Secondly, sponsored search auctions hosted by search engines (Google, Yahoo!, MSN and the like) where advertisers bid on keywords include budget specification as a feature. A common (and natural) assumption in keyword auctions that is typically made is that bids of advertisers are much smaller than the budgets. However, with the extension of the sponsored search medium from the web onto the more classical media, such as radio and television[1] where this assumption is not as reasonable, the general budget-constrained auctions need to be addressed.

Previous and Related Work: MBA is known to be NP-hard as it encodes PARTITION [2], and this observation was made concurrently by many authors ([24, 54, 2, 45]). The first approximation algorithm for the problem was given by Garg, Kumar and Pandit[24] who gave a $2/(1+\sqrt{5})(\simeq 0.618)$ factor approximation. Andelman and Mansour[2] improved the factor to $(1-1/e)(\simeq 0.632)$. For the special case when budgets of all bidders were equal, [2] improved the factor to 0.717. Very recently, and independent of our work, Azar et.al. [3] obtained a 2/3-factor for the general MBA problem. They also considered a *uniform* version of the problem where for every item j, the bid of any agent is either b_j (independent of the agent) or 0. They gave a $1/\sqrt{2}(\simeq 0.707)$ factor for the same. All these algorithms are based on a natural LP relaxation (LP(1) in Section 2.0.4) which we use as well.

In the setting of sponsored search auctions, MBA, or rather β-MBA with $\beta \to 0$, has been studied mainly in an online context. Mehta et.al.[48] and later, Buchbinder et.al.[9] gave $(1-1/e)$-competitive algorithms when the assumption of bids being

[1] see for instance `http://www.google.com/adwords/audioads/` and `http://www.google.com/adwords/tvads`

[2] PARTITION: Given n integers $a_1, \cdots, a_n$ and a target B, decide whether there is a subset of these integers adding up to exactly B

small to budget is made. The dependence of the factor on β is not quite clear from either of the works. Moreover, as per our knowledge, nothing better was known the approximability of the *offline* β-MBA than what was suggested by algorithms for MBA.

Our results: We give two approximation algorithms for MBA. The first, based on iterative LP rounding, attains a factor of 3/4. The algorithm described in Section 2.1. The second algorithm, based on the primal-dual schema, is faster and attains a factor of $3/4(1-\epsilon)$, for any $\epsilon > 0$. The running time of the algorithm is $\tilde{O}(\frac{nm}{\epsilon})^3$, and is thus almost linear for constant ϵ and dense instances. We describe the algorithm in Section 2.2. Our algorithms can be extended suitably for β-MBA as well giving a $1 - \beta/4$ factor approximation algorithm.

In Section 2.3, we show it is NP hard to approximate MBA to a factor better than 15/16 via a gap-preserving reduction from Max-3-Lin(2). Our hardness instances are *uniform* in the sense of Azar et.al. [3] implying uniform MBA is as hard. Our hardness reductions extend to give a $(1 - \beta/16)$ hardness for β-MBA as well. Interestingly, our reductions can be used to obtain better inapproximability results for other problems: SWM (15/16 hardness even with demand queries), GAP (10/11 hardness) and MSSF(10/11 and 13/14 for the edge and node weighted versions), which we elaborate below.

2.0.3 Relations to other allocation problems

Submodular Welfare Maximization (SWM): As in the definition of MBA, let Q be a set of m indivisible items and A be a set of n agents. For agent i, let $u_i : 2^Q \to \mathbb{R}_+$ be a utility function where for a subset of items $S \subseteq Q$, $u_i(S)$ denote the utility obtained by agent i when S is allocated to it. Given an allocation of items

[3] the ˜ hides logarithmic factors

to agents, the total *social welfare* is the sum of utilities of the agents. The *welfare maximization* problem is to find an allocation of maximum social welfare. We assume the existence of a *demand oracle*: for any agent i and prices $p_1, p_2, \cdots, p_m$ for all items in Q, returns a subset $S \subseteq Q$ which maximizes $(u_i(S) - \sum_{j \in S} p_j)$.

Welfare maximization problems have been extensively studied (see, for example, [7]) in the past few years with various assumptions made on the utility functions. One important set of utility functions are *monotone submodular utility functions*. A utility function u_i is submodular if for any two subsets S, T of items, $u_i(S \cup T) + u_i(S \cap T) \leq u_i(S) + u_i(T)$. The welfare maximization problem when all the utility functions are submodular is called the submodular welfare maximization problem or simply SWM. Feige and Vondrak [20] gave an $(1 - 1/e + \rho)$-approximation for SWM with $\rho \sim 0.0001$ and showed that it is NP-hard to approximate SWM to better than $275/276$.[4]

MBA is a special case of SWM. This follows from the observation that the utility function $u_i(S) = \min(B_i, \sum_{j \in S} b_{ij})$ when B_i, b_{ij}'s are fixed is a submodular function. In Section 2.3.1, we show that in the hardness instances of MBA, the demand oracle can be simulated in poly-time and therefore the 15/16 hardness of approximation for MBA implies a 15/16-hardness of approximation for SWM as well.

Generalized Assignment Problem (GAP): GAP is a problem quite related to MBA: Every item j, along with the bid (profit) b_{ij} for agent (bin) i, also has an inherent size s_{ij}. Instead of a budget constraint, each agent (bin) has a capacity constraint C_i which defines feasible sets: A set S is feasible for (bin) i if $\sum_{j \in S} s_{ij} \leq C_i$. The goal is to find a revenue (profit) maximizing feasible assignment. The main difference between GAP and MBA is that in GAP we are *not allowed* to violate capacity constraints, while in MBA the budget constraint only caps the revenue. As

[4] We remark that SWM with a different oracle, the *value oracle*, has recently been resolved. There was a $(1 - 1/e)$ hardness given by Khot et.al.[38] and recently Vondrak[64] gave a matching polynomial time algorithm.

was noted by Chekuri and Khanna[14], a 1/2 approximation algorithm was implicit in the work of Shmoys and Tardos[57]. The factor was improved by Fleischer et.al.[23] to $1 - 1/e$. In the same paper [20] where they give the best known algorithm for SWM, Feige and Vondrák[20] also give a $(1 - 1/e + \rho')$ algorithm for GAP ($\rho' \leq 10^{-5}$). The best known hardness for GAP was $1 - \epsilon$, for some small ϵ which was given by Chekuri and Khanna [14] via a reduction from maximum 3D-matching. Improved hardness results for maximum 3D matching by Chlebik and Chlebikova[16], imply a 422/423 hardness for GAP.

Although MBA and GAP are in some sense incomparable problems, we can use our hardness techniques to get a 10/11 factor hardness of approximation for GAP in Section 2.3.3.

Maximum Spanning Star-Forest Problem (MSSF): Given an undirected unweighted graph G, the MSSF problem is to find a forest with as many edges such that each tree in the forest is a star - all but at most one vertex of the tree are leaves. The edge-weighted MSSF is the natural generalization with weights on edges. The node-weighted MSSF has weights on vertices and the weight of a star is the weight on the leaves. If the star is just an edge, then the weight of the star is the maximum of the weights of the end points.

The unweighted and edge-weighted MSSF was introduced by Nguyen et.al [50] who gave a 3/5 and 1/2-approximation respectively for the problems. They also showed APX hardness of the unweighted version. Chen et.al. [15] improved the factor of unweighted MSSF to 0.71 and introduced node-weighted MSSF giving a 0.64 factor algorithm for it. They also give a 31/32 and 19/20 hardness for the node-weighted and edge-weighted MSSF problems.

Although, at the face of it, MSSF does not seem to have a relation with MBA, once again our hardness technique can be used to improve the hardness of node-weighted

and edge-weighted MSSF to 13/14 and 10/11, respectively.

2.0.4 The LP Relaxation for MBA

One way to formulate MBA as an integer program is the following:

$$\max\{\sum_{i\in A}\pi_i :\ \pi_i = \min(B_i, \sum_{j\in Q} b_{ij}x_{ij}),\ \forall i;\quad \sum_{i\in A} x_{ij} \leq 1, \forall j;\quad x_{ij} \in \{0,1\}\ \}$$

Relaxing the integrality constraints to non-negativity constraints gives an LP relaxation for the problem. We work with the following equivalent LP relaxation of the problem. The equivalence follows by noting that in there exists an optimal fractional solution, $B_i \geq \sum_{j\in Q} b_{ij}x_{ij}$. This was noted by Andelman and Mansour[2] and a similar relaxation was used by Garg et.al. [24].

$$\max\{\sum_{i\in A, j\in Q} b_{ij}x_{ij} :\quad \forall i \in A,\ \sum_{j\in Q} b_{ij}x_{ij} \leq B_i;\quad \forall j \in Q,\ \sum_{i\in A} x_{ij} \leq 1;$$
$$\forall i \in A, j \in Q,\ x_{ij} \geq 0\} \tag{1}$$

We remark that the assumption $b_{ij} \leq B_i$ is crucial for this LP to be of any use. Without this assumption it is easy to construct examples having arbitrarily high integrality gaps. Consider the instance with one item, n agents each having a budget 1 but bidding n on the item. The LP has a solution of value n while the maximum welfare is obviously 1.

2.0.5 Technical Contributions

Apart from being an important problem in its own right, we believe the MBA problem is interesting as it allows us to enhance certain algorithmic ideas. As we mention above, one of our algorithms is an iterative rounding algorithm using LP(1). Recently, the technique of iterative rounding has achieved considerable success in designing approximation algorithms [35, 58, 44]. However, these successes have been limited

to minimization problems, and as per our knowledge, this work is the first iterative rounding based result for a natural maximization problem.

The iterative rounding schema can be broadly described in the following two step procedure: find an optimal solution satisfying some *desirable property* (for instance, some variable has value 1 or at least 1/2 ([35])), then move to a *residual problem* after rounding some variables to 1 and iterate. The proof of approximation factor, say $\alpha > 1$, is shown by proving that the cost incurred by the algorithm in one iteration is at most α times the drop in the LP-value across the iteration.

In minimization problems, the constraints on the variables are normally covering constraints. Thus, a solution to the original problem is also a solution to the natural residual problem obtained after rounding up. The non-trivial part of most algorithms ([35, 58, 44]) lies in proving the existence of a *desirable property.*

In this problem, the desirable property as we show later (and this has been known for some time) is that $x_{ij} = 1$ for some agent i and item j. However, in the natural residual problem obtained after the item j is allocated to agent i, the LP might drop by as much as *twice* as the value obtained by the algorithm (giving only a 1/2 factor). To overcome this difficulty, we show how to move to a *residual problem* in a non-trivial manner which allows us to control the LP-drop across iterations. We give the details in Section 2.1. We believe this technique of a clever definition of the residual problem might be an approach towards iterative rounding of other maximization problems like SWM and GAP. That said, it is also non-trivial for such problems even to get the desirable property of a solution; in fact it is unclear what a desirable property should be.

We also give a primal-dual algorithm for MBA in Section 2.2 which for any $\epsilon > 0$ gives a $\frac{3}{4}(1 - \epsilon)$ approximation factor. Primal-dual algorithms have been successful in obtaining approximation algorithms for minimization problems (whose duals are

maximization). The typical schema is to start with an all-zeroes feasible dual solution and then raising the duals till some dual constraint goes tight which suggests the primal variable to pick. Since MBA is a maximization problem, the dual is a minimization and the all-zeroes solution is not feasible any more. We deviate from the schema by setting a *subset* of dual variables to zero and raising these variables. The remaining dual variables are set so as to maintain dual feasibility. We believe that our method of identifying the correct dual variables to work with might be applicable for other maximization problems as well.

2.1 An iterative rounding algorithm for MBA

Let $\mathcal{P}$ be a problem instance defined by the bids and budgets of every agent, that is $\mathcal{P} := (\{b_{ij}\}_{i,j}, \{B_i\}_i)$. With $\mathcal{P}$, we associate a bipartite graph $G(\mathcal{P}) = (A \cup Q, E)$, with $(i, j) \in E$ if i bids on j.

Let $x^*(\mathcal{P})$ be an extreme point solution to LP(1) for the problem instance $\mathcal{P}$. For brevity, we omit writing the dependence on $\mathcal{P}$ when the instance is clear from context. Let E^* be the support of the solution, that is $E^* := \{(i, j) \in E : x^*_{ij} > 0\}$. Note that these are the only *important* edges - one can discard all bids of agents i on item j when $(i, j) \notin E^*$. This does not change the LP optimum and a feasible integral solution in this instance is a feasible solution of the original instance. Call the set of neighbors of an agent i in $G[E^*]$ as $\Gamma(i)$. Call an agent *tight* if $\sum_j b_{ij} x^*_{ij} = B_i$.

The starting point of the algorithm is the following claim about the structure of the extreme point solution. Such an argument using polyhedral combinatorics, was first used in the machine scheduling paper of Lenstra, Shmoys and Tardos [46]. A similar claim can be found in the thesis of Andelman [1].

Claim 2.1.1 *The graph, $G[E^*]$, induced by E^* can be assumed to be a forest. Moreover, except for at most one, all the leaves of a connected component are items. Also at most one agent in a connected component can be* non-tight.

Proof: Consider the graph $G[E^*]$. Without loss of generality assume that it is a single connected component. Otherwise we can treat every connected component as a separate instance and argue on each of them separately. Thus, $G[E^*]$ has $(n+m)$ nodes. Also since there are $(n+m)$ constraints in the LP which are not non-negativity constraints, therefore support of any extreme point solution can be of size atmost $(n+m)$. This follows from simple polyhedral combinatorics: at an extreme point, the number of inequalities going tight is at least the number of variables. Since there are only $(n+m)$ constraints which are not non-negativity constraints, all but at most $(n+m)$ variables must satisfy the non-negativity constraints with equality, that is, should be 0. Thus $|E^*| \leq n+m$.

Hence there is at most one cycle in $G[E^*]$. Suppose the cycle is: $(i_1, j_1, i_2, j_2, \cdots, j_k, i_1)$, where $\{i_1, \cdots, i_k\}$ and $\{j_1, \cdots, j_k\}$ are the subsets of agents and items respectively. Consider the feasible fractional solution obtained by decreasing x^* on (i_1, j_1) by ϵ_1 and increasing on (i_2, j_1) by ϵ_1, decreasing on (i_2, j_1) by ϵ_2 , increasing on (i_2, j_1) by ϵ_2, and so on. Note that if the ϵ_i's are small enough, the item constraints are satisfied. The relation between ϵ_1 and ϵ_2 (and cascading to other ϵ_r's) is: $\epsilon_1 b_{i_2,j_1} = \epsilon_2 b_{i_2,j_2}$, that is, the fraction of money spent by i_2 on j_1 equals the money freed by j_2. The exception is the last ϵ_k, which might not satisfy the condition with ϵ_1. If $\epsilon_k b_{i_1,j_k} > \epsilon_1 b_{i_1,j_1}$, then just stop the increase on the edge (i_1, j_k) to the point where there is equality. If $\epsilon_k b_{i_1,j_k} < \epsilon_1 b_{i_1,j_1}$, then start the whole procedure by increasing x^* on the edge (i_1, j_1) instead of decreasing and so on. In one of the two cases, we will get a feasible solution of equal value and the ϵ_i's can be so scaled so as to reduce x^* on one edge to 0. In other words, the cycle is broken without decreasing the LP value.

Thus, $G[E^*]$ is a tree. Moreover, since $(n+m-1)$ edges are positive, there must be $(n+m-1)$ equalities among the budget and the item constraints. Thus at most one budget constraint can be violated which implies at most one agent can *non tight*. Now since the bids are less than the budget, therefore if an agent is a leaf of the tree

$G[E^*]$ then he must be *non-tight*. Hence atmost one agent can be a leaf of the tree $G[E^*]$.

□

Call an item a *leaf item*, if it is a leaf in $G[E^*]$. Also call an agent i a *leaf agent* if, except for at most one, all of his neighboring items in $E^*(\mathcal{P})$ are leaves. Note the above claim implies each connected component has at least one leaf item and one leaf agent: in any tree there are two leaves both of which cannot be agents, and there must be an agent with all but one of its neighbors leaves and thus leaf items. For the sake of understanding, we first discuss the following natural iterative algorithm which assigns the leaf items to their neighbors and then adjusts the budget and bids to get a new *residual problem*.

1/2-approx algorithm: Solve $LP(\mathcal{P})$ to get $x^*(\mathcal{P})$. Now pick a leaf agent i. Assign all the leaf items in $\Gamma(i)$ to i. Let j be the unique non-leaf item (if any) in $\Gamma(i)$. Form the new instance $\mathcal{P}'$ by removing $\Gamma(i) \setminus j$ and all incident edges from $\mathcal{P}'$. Let $b = \sum_{l \in \Gamma(i) \setminus j} b_{il}$, be the portion of budget spent by i. Now modify the budget of i and his bid on j in $\mathcal{P}'$ as follows: $B'_i := B_i - b$, and $b'_{ij} := \min(b_{ij}, B'_i)$. Its instructive to note the drop $(b_{ij} - b'_{ij})$ is at most b. (We use here the assumption bids are always smaller than budgets). Iterate on the instance $\mathcal{P}'$.

The above algorithm is a 1/2-approximation algorithm. In every iteration, we show that the revenue generated by the items allocated is at least 1/2 of the drop in the LP value $(LP(\mathcal{P}) - LP(\mathcal{P}'))$. Suppose in some iteration, i be the leaf agent chosen, and let j be the its non-leaf neighbor, and let the revenue generated by algorithm be b. Note that x^*, the solution to $\mathcal{P}$, restricted to the edges in $\mathcal{P}'$ is still a feasible solution. Thus the drop in the LP is: $b + (b_{ij} - b'_{ij})x_{ij}$. Since $(b_{ij} - b'_{ij})$ is atmost b, and x_{ij} at most 1, we get $LP(\mathcal{P}) - LP(\mathcal{P}') \leq 2b$.

To prove a better factor in the analysis, one way is to give a better bound on the drop, $(LP(\mathcal{P}) - LP(\mathcal{P}'))$. Unfortunately, the above analysis is almost tight and there exists an example where the LP drop in the first iteration is $\simeq$ twice the revenue generated by the algorithm in that iteration. Thus, for an improved analysis for this algorithm, one needs a better amortized analysis across different iterations rather than analyzing iteration-by-iteration. This seems non-trivial as we solve the LP again at each iteration and the solutions could be very different across iterations making it harder to analyze over iterations.

Instead, we modify the above algorithm by defining the *residual problem* $\mathcal{P}'$ in an non-trivial manner. After assigning leaf items to agent i, we do not decrease the budget by the amount assigned, but keep it a little "larger". Thus these agents *lie* about their budgets in the subsequent rounds, and we call these *lying* agents. Since the budget doesn't drop too much, the LP value of the residual problem doesn't drop much either. A possible trouble would arise when items are being assigned to lying agents since they do not pay as much as they have bid. This leads to a trade-off and we show by suitably modifying the residual problem one can get a 3/4 approximation. We now elaborate.

Given a problem instance $\mathcal{P}_0 := \mathcal{P}$, the algorithm proceeds in stages producing newer instances at each stage. On going from $\mathcal{P}_i$ to $\mathcal{P}_{i+1}$, at least one item is allocated to some agent. Items are never de-allocated, thus the process ends in at most m stages. The *value* of an item is defined to be the payment made by the agent who gets it. That is, $value(j) = \min(b_{ij}, B_i - spent(i))$, where $spent(i)$ is the value of items allocated to i at the time j was being allocated. We will always ensure the condition that a lying agent i bids on at most one item j. We will call j the *false item* of i and the bid of i on j to be i's *false bid*. In the beginning no agent is lying.

We now describe the k-th iteration of the iterative algorithm which we call MBA-ITER (Algorithm 1).

Algorithm 1 k-th step of MBA-ITER

1. Solve $LP(\mathcal{P}_k)$. Remove all edges which are not in $E^*(\mathcal{P}_k)$. These edges will stay removed in all subsequent steps.

2. If there is a lying agent i with $x^*_{ij} = 1$ for his false item j, assign item j to him. In the next instance, $\mathcal{P}_{k+1}$, remove i and j. Proceed to $(k+1)$-th iteration.

3. If there is a non-lying agent i such that all the items in $\Gamma(i)$ are leaf items. Then allocate $\Gamma(i)$ to i. Remove i, $\Gamma(i)$ and all the incident edges to get the new instance $\mathcal{P}_{k+1}$ and proceed to $(k+1)$-th iteration step.

4. Pick a *tight* leaf agent i. Notice that i must have at least two items in $\Gamma(i)$, otherwise tightness would imply that the unique item is a leaf item and thus either step 2 or step 3 must have been performed. Moreover, exactly one item in $\Gamma(i)$ is not a leaf item, and let j be this unique non-leaf item. Allocate all the items in $\Gamma(i) \setminus j$ to i. In $\mathcal{P}_{k+1}$, remove $\Gamma(i) \setminus j$ and all incident edges. Also, modify the budget and bids of agent i. Note that agent i now bids *only* on item j as there are no other edges incident to i. Let the new bid of agent i on item j be
$$b'_{ij} := \max(0, \frac{4b_{ij}x^*_{ij} - B_i}{3x^*_{ij}})$$
Let the new budget of agent i be $B'_i := b'_{ij}$. Call i lying and j be his false item. Proceed to $(k+1)$-th iteration.

Claim 2.1.2 *In each step, at least one item is allocated and thus* MBA-ITER *terminates in m steps.*

Proof: We show that one of the three steps 2,3 or 4 is always performed and thus some item is always allocated. Consider any component. If a component has only one agent i, then all the items in $\Gamma(i)$ are leaf items. If $\Gamma(i)$ has more than two items, then the agent cannot be lying since the lying agent bids on only one item and Step 3 can be performed. If $\Gamma(i) = \{j\}$, then $x^*_{ij} = 1$ since otherwise x^*_{ij} could be increased giving a better solution. Thus Step 2 or 3 can always be performed depending on if i is lying or not. If the component has at least two agents, then it must have two leaf

agents. This can be seen by rooting the tree at any item. At least one of them, say i, is tight by Claim 2.1.1. Thus Step 4 can be performed. □

Theorem 2.1.3 *Given a problem instance $\mathcal{P}$, the allocation obtained by algorithm* MBA-Iter *attains value at least $\frac{3}{4} \cdot LP(\mathcal{P})$.*

Proof: Let $\Delta_k := LP(\mathcal{P}_k) - LP(\mathcal{P}_{k+1})$ denote the drop in the optimum across the k-th iteration. Denote the set of items allocated at step k as Q_k. Note that the total value of the algorithm is $\sum_{j \in Q} value(j) = \sum_k (\sum_{j \in Q_k} value(j))$. Also, the LP optimum of the original solution is $LP(\mathcal{P}) = \sum_k \Delta_k$ since after the last item is allocated the LP value becomes 0. The following lemma proves the theorem. □

Lemma 2.1.4 *In every stage k, $value(Q_k) := \sum_{j \in Q_k} value(j) \geq \frac{3}{4}\Delta_k$.*

Proof: Items are assigned in either Step 2,3 or 4. Let us analyze Step 2 first. Let i be the lying agent obtaining his false item j. Since $x^*_{ij} = 1$ and lying agents bid on only one item, the remaining solution (keeping the same x^* on all remaining edges) is a valid solution for the LP in $\mathcal{P}_{k+1}$. Thus

$$LP(\mathcal{P}_k) - LP(\mathcal{P}_{k+1}) \leq b',$$

where b' is the false bid of lying agent i on item j. Let b be the bid of agent i on item j, before it was made lying. Then, from Step 4 we know that $b' := \frac{4bx - B}{3x}$, where x was the fraction of item j assigned to i and B is the budget of i. Moreover, the portion of budget spent by i is at most $(B - bx)$. This implies $value(j) \geq bx$. The claim follows by noting for all $b \leq B$ and all x,[5]

$$bx \geq \frac{3}{4} \cdot \frac{4bx - B}{3x}$$

In Step 3, in fact the LP drop equals the value obtained - both the LP drop and the value obtained is the sum of bids on items in $\Gamma(i)$ or B_i, whichever is less.

[5] $4bx^2 - 4bx + B = b(2x-1)^2 + (B-b) \geq 0$

Coming to step 4, $Q_k = \Gamma(i) \setminus j$ be the set of goods assigned to the tight, non-lying leaf agent i. Let b and b' denote the bids of i on j before and after the step: b_{ij} and b'_{ij}. Let x be x^*_{ij}. Note that $x^*_{il} \leq 1$ for all $l \in Q_k$. Also, x^* restricted to the remaining goods still is a feasible solution in the modified instance $\mathcal{P}_{k+1}$. Since the bid on item j changes from b to b', the drop in the optimum is at most

$$LP(\mathcal{P}_k) - LP(\mathcal{P}_{k+1}) \leq (\sum_{l \in Q_k} b_{il}) + (bx - b'x)$$

Note that $value(Q_k) = \sum_{l \in Q_k} b_{il} \geq B - bx$ by tightness of i. We now show $(bx - b'x) \leq \frac{1}{3} \cdot value(Q_k)$ which would prove the lemma.

If $b' = 0$, this means $4bx \leq B$. Thus, $value(Q_k) \geq B - bx \geq 3bx$. Otherwise, we have

$$(bx - b'x) = bx - \frac{4bx - B}{3x} \cdot x = \frac{B - bx}{3} \leq value(Q_k)/3$$

implying the claim, as before. □

Remark: The above algorithm can be extended to the case of β-MBA as well. The only difference is in the modification of bids and budgets in Step 4, where b'_{ij} will now be set to $\frac{4b_{ij}x^*_{ij} - \beta \cdot B_i}{(4-\beta)x^*_{ij}}$. It is not hard to modify Theorem 2.1.3 to show that this gives a $1 - \beta/4$-factor algorithm for β-MBA.

2.2 Primal-dual algorithm for MBA

In this section we give a faster primal-dual algorithm for MBA although we lose a bit on the factor. The main theorem of this section is the following:

Theorem 2.2.1 *For any $\epsilon > 0$, there exists an algorithm which runs in $\tilde{O}(nm/\epsilon)$ time and gives a $\frac{3}{4} \cdot (1 - \epsilon)$-factor approximation algorithm for MBA.*

Let us start by taking the dual of the LP relaxation LP(1).

$$DUAL :=$$
$$\min\{\sum_{i\in A} B_i\alpha_i + \sum_{j\in Q} p_j : \ \forall i \in A, j \in Q;\ p_j \geq b_{ij}(1-\alpha_i);\ \forall i \in A, j \in Q;\ p_j, \alpha_i \geq 0\} \tag{2}$$

We make the following interpretation of the dual variables: Every agent retains α_i of his budget, and all his bids are modified to $b_{ij}(1-\alpha_i)$. The price p_j of a good is the highest modified bid on it. The dual program finds retention factors to minimize the sum of budgets retained and prices of items. We start with a few definitions.

Definition 2 *Let $\Gamma : A \to 2^Q$ be an allocation of items to agents and let the set $\Gamma(i)$ be called the items* owned *by i. Let $S_i := \sum_{j\in\Gamma(i)} b_{ij}$ denote the total bids of i on items in $\Gamma(i)$. Note that the revenue generated by Γ from agent i is $\min(S_i, B_i)$. Given α_i's, the* prices generated by Γ *is defined as follows: $p_j = b_{ij}(1-\alpha_i)$, where j is owned by i. Call an item* wrongly allocated *if $p_j < b_{lj}(1-\alpha_l)$ for some agent l, call it* rightly allocated *otherwise. An allocation Γ is called* valid *(w.r.t α_i's) if all items are rightly allocated, that is, according to the interpretation of the dual given above, all items go to agents with the highest modified bid ($b_{ij}(1-\alpha_i)$) on it. Note that if Γ is valid, (p_j,α_i)'s form a valid dual solution. Given an $\epsilon > 0$, Γ is ϵ-*valid *if $p_j/(1-\epsilon)$ satisfies the dual feasibility constraints with the α_i's.*

Observe that given α_i's; and given an allocation Γ and thus the prices p_j generated by it, the objective of the dual program can be treated agent-by-agent as follows

$$DUAL = \sum_i Dual(i), \quad \text{where } Dual(i) = B_i\alpha_i + \sum_{j\in\Gamma(i)} p_j = B_i\alpha_i + S_i(1-\alpha_i) \tag{3}$$

Now we are ready to describe the main idea of the primal-dual schema. The algorithm starts with all α_i's set to 0 and an allocation valid w.r.t to these. We will "pay-off" this dual by the value obtained from the allocation agent-by-agent. That

is, we want to pay-off $Dual(i)$ with $\min(B_i, S_i)$ for all agents i. Call an agent *paid for* if $\min(B_i, S_i) \geq \frac{3}{4} Dual(i)$. We will be done if we find α_i's and an allocation valid w.r.t these such that all agents are paid for.

Let us look at when an agent is paid for. From the definition of $Dual(i)$, an easy calculation shows that an agent is paid for iff $S_i \in [L(\alpha_i), U(\alpha_i)] \cdot B_i$, where $L(\alpha) = \frac{3\alpha}{1+3\alpha}$ and $U(\alpha) = \frac{4-3\alpha}{3-3\alpha}$. Note that S_i depends on Γ which was chosen to be valid w.r.t. α_i's. Moreover, observe that increasing α_i can only lead to the decrease of S_i and vice-versa. This suggests the following next step: for agents i which are unpaid for, if $S_i > U(\alpha_i)B_i$, increase α_i and if $S_i < L(\alpha_i)B_i$, decrease α_i and modify Γ to be the valid allocation w.r.t the α_i's.

However, it is hard to analyze the termination of an algorithm which both increases and decreases α_i's. This is where we use the following observation about the function $L()$ and $U()$. (In fact 3/4 is the largest factor for which the corresponding $L()$ and $U()$ have the following property; see Remark 2.2.4).

Property 2.2.2 *For all α, $U(\alpha) \geq L(\alpha) + 1$.* [6]

The above property shows that an agent with $S_i > U(\alpha_i)B_i$ on losing a *single item* j will still have $S_i > U(\alpha_i)B_i - b_{ij} \geq (U(\alpha_i) - 1)B_i \geq L(\alpha_i)B_i$, for any $\alpha_i \in [0, 1]$. Also observe that in the beginning when α_i's are 0, $S_i \geq L(\alpha_i)B_i$. Thus if we can make sure that the size of $\Gamma(i)$ decreases by at most one, when the α_i's of an unpaid agent i is increased, then the case $S_i < L(\alpha_i)B_i$ never occurs and therefore we will never have to decrease α's and termination will be guaranteed.

However, an increase in α_i can lead to movement of more than one item from the current allocation of agent i to the new valid allocation. Thus to ensure *steady* progress is made throughout, we move to ϵ-valid allocations and get a $\frac{3}{4} \cdot (1 - \epsilon)$ algorithm. We now give details of the Algorithm 2.

[6] $U(\alpha) - 1 = \frac{1}{3-3\alpha} \geq \frac{3\alpha}{1+3\alpha} =: L(\alpha) \Leftarrow 1 + 3\alpha \geq 9\alpha(1-\alpha) \Leftarrow 9\alpha^2 - 6\alpha + 1 \geq 0$

Algorithm 2 MBA-PD: Primal Dual Algorithm for MBA

Define $\epsilon_i := \epsilon \cdot \frac{1-\alpha_i}{\alpha_i}$. Throughout, p_j will be the price generated by Γ and current α_i's.

1. Initialize $\alpha_i = 0$ for all agents. Let Γ be the allocation assigning item j to agent i which maximizes b_{ij}.

2. Repeat the following till all agents are paid for:

   Pick an agent i who is not paid for (that is $S_i > U(\alpha_i)B_i$), arbitrarily. Repeat till i becomes paid for:

   If i has no wrongly allocated items in $\Gamma(i)$, then increase $\alpha_i \rightarrow \alpha_i(1+\epsilon_i)$. (Note that when $\alpha_i = 0$, ϵ_i is undefined. In that case, modify $\alpha_i = \epsilon$ from 0.)

   Else pick any one wrongly allocated item j of agent i, and modify Γ by allocating j to the agent l who maximizes $b_{lj}(1 - \alpha_l)$. (Note that this makes j rightly allocated but can potentially make agent l not paid for).

Claim 2.2.3 *Throughout the algorithm, $S_i \geq L(\alpha_i)B_i$.*

Proof: The claim is true to start with ($L(0) = 0$). Moreover, S_i of an agent i decreases *only if* i is not paid for, that is, $S_i > U(\alpha_i)B_i$. Now, since items are transferred one at a time and each item can contribute at most B_i to S_i, the fact $U(\alpha) \geq 1 + L(\alpha)$ for all α proves the claim. □

Remark 2.2.4 *In general, one can compare Dual(i) and* $\min(S_i, B_i)$ *to figure out what L, U should be to get a ρ-approximation. As it turns out, the largest ρ for which U, L satisfies property 2.2.2 is* $3/4$ *(and it cannot be any larger due to the integrality gap example). However, the bottleneck above is the fact that each item can contribute*

at most B_i to S_i. Note that in the case of β-MBA this is $\beta \cdot B_i$ and indeed this is what gives a better factor algorithm. Details in Section 2.2.1.

Theorem 2.2.5 *For any $\epsilon > 0$, given α_i's, an allocation Γ ϵ-valid w.r.t it and p_j, the prices generated by Γ; if all agents are paid for then Γ is a $3/4(1-\epsilon)$-factor approximation for MBA.*

Proof: Consider the dual solution (p_j, α_i). Since all agents are paid for, $\min(B_i, S_i) \geq 3/4 \cdot Dual(i)$. Thus the total value obtained from Γ is at least $3/4 \sum_{i \in A} Dual(i)$. Moreover, since Γ is ϵ-valid, $(p_j/(1-\epsilon), \alpha_i)$ forms a valid dual of cost $\frac{1}{1-\epsilon} \sum_{i \in A} Dual(i)$ which is an upper bound on the optimum of the LP and thus the proof follows. □

Along with Theorem 2.2.5, the following theorem about the running time proves Theorem 2.2.1.

Theorem 2.2.6 *Algorithm* MBA-PD *terminates in $(nm \cdot \ln(3m)/\epsilon)$ iterations with an allocation Γ with all agents paid for. Moreover, the allocation is ϵ-valid w.r.t the final α_i's.*

Proof: Let us first show the allocation throughout remains ϵ-valid w.r.t. the α_i's. Note that initially the allocation is valid. Subsequently, the price of an item j generated by Γ decreases only when the α_i of an agent i owning j increases. This happens only in Step 2, and moreover j must be rightly allocated before the increase. Now the following calculation shows that after the increase of α_i, p_j decreases by a factor of $(1-\epsilon)$. Thus, $(p_j/(1-\epsilon), \alpha_i)$'s form a valid dual solution implying Γ is ϵ-valid.

$$\begin{aligned} p_j^{(new)} &= b_{ij}(1 - \alpha_i^{(new)}) = b_{ij}(1 - \alpha_i(1+\epsilon_i)) \\ &= b_{ij}(1-\alpha_i)(1 - \epsilon_i\alpha_i/(1-\alpha_i)) = p_j^{(old)}(1-\epsilon) \end{aligned}$$

Now in Step 2, note that until there are agents not paid for, either we decrease the number of wrongly allocated items or we increase the α_i for some agent i. That

is, in at most m iterations of Step 2, α_i of some agent becomes $\alpha_i(1+\epsilon_i)$. Now, note that if $\alpha_i > 1 - 1/3m$ for some agent, he is paid for. This follows simply by noting that $S_i \leq mB_i = U(1-1/3m) \cdot B_i$ and the fact that $S_i \geq L(\alpha_i)B_i$, for all α_i.

Claim 2.2.7 *If α_i is increased $t > 0$ times, then it becomes $1-(1-\epsilon)^t$.*

Proof: At $t = 1$, the claim is true as α_i becomes ϵ. Suppose the claim is true for some $t \geq 1$. On the $t+1$th increase, α_i goes to

$$\begin{aligned} \alpha_i(1+\epsilon_i) &= \alpha_i + \epsilon(1-\alpha_i) = \alpha_i(1-\epsilon)+\epsilon \\ &= (1-(1-\epsilon)^t)(1-\epsilon)+\epsilon \\ &= 1-(1-\epsilon)^{t+1} \end{aligned}$$

□

Thus if α_i is increased $\ln(3m)/\epsilon$ times, i becomes paid for throughout the remainder of the algorithm. Since there are n agents, and in each m-steps some agent's α_i increases, in $(nm \cdot \ln(3m)/\epsilon)$ iterations all agents are paid for and the algorithm terminates. □

2.2.1 Extension to β-MBA

The algorithm for β-MBA is exactly the same as Algorithm 2. The only difference is the definition of *paid for* and $L(), U()$. Call an agent paid for if $\min(B_i, S_i) \geq \frac{4-\beta}{4} Dual(i)$. Define the function $L(\alpha) := \frac{\alpha(4-\beta)}{\alpha(4-\beta)+\beta}$ and $U(\alpha) := \frac{(1-\alpha)(4-\beta)+\beta}{(1-\alpha)(4-\beta)}$ Note that when $\beta = 1$, the definitions coincide with the definitions in the previous section.

Claim 2.2.8 *Given α_i's, agent is paid for if $S_i \in [L(\alpha_i), U(\alpha_i)] \cdot B_i$*

Proof: Agent i is paid for if both $B_i \geq \frac{(4-\beta)}{4}(B_i\alpha_i + S_i(1-\alpha_i))$ and $S_i \geq \frac{(4-\beta)}{4}(B_i\alpha_i + S_i(1-\alpha_i))$. Let us lose the subscript for the remainder of the proof.
The first implies

$$S(1-\alpha) \leq B(\frac{4}{(4-\beta)} - \alpha) \Rightarrow S(1-\alpha) \leq B\frac{(4-\beta)(1-\alpha)+\beta}{(4-\beta)} \Rightarrow S \leq U(\alpha)B$$

The second implies

$$S(\frac{4}{(4-\beta)} - (1-\alpha)) \geq B\alpha \Rightarrow S\frac{\alpha(4-\beta)+\beta}{(4-\beta)} \geq B\alpha \Rightarrow S \geq L(\alpha)B$$

□

Property 2.2.9 *For all α, $U(\alpha) \geq L(\alpha) + \beta$*

Proof: Note that $U(\alpha) = 1 + \frac{\beta}{(1-\alpha)(4-\beta)}$ and $L(\alpha) = 1 - \frac{\beta}{\alpha(4-\beta)+\beta}$. Now,

$$\begin{aligned}
U(\alpha) - \beta \geq \quad L(\alpha) \Leftarrow \quad & \frac{\beta}{(1-\alpha)(4-\beta)} - \beta \;\geq\; \frac{\beta}{\alpha(4-\beta)+\beta} \\
\Leftarrow \quad & \frac{1}{(1-\alpha)(4-\beta)} \geq \frac{\alpha(4-\beta)-(1-\beta)}{\alpha(4-\beta)+\beta} \\
\Leftarrow \quad & \alpha(4-\beta)+\beta \quad \geq \quad (4-\beta)^2\alpha(1-\alpha) - (1-\alpha)(1-\beta)(4-\beta) \\
\Leftarrow \quad & \alpha^2(4-\beta)^2 - \alpha(4-\beta)((1-\beta)+(4-\beta)-1) + \beta + (1-\beta)(4-\beta) \geq 0 \\
\Leftarrow \quad & (\alpha(4-\beta))^2 - 2\alpha(4-\beta)(2-\beta) + (2-\beta)^2 \geq 0 \\
\Leftarrow \quad & (\alpha(4-\beta) - (2-\beta))^2 \geq 0
\end{aligned}$$

which is true for any α. □

Theorem 2.2.10 *The algorithm 2 with the above definitions gives a $(1-\beta/4)(1-\epsilon)$-factor approximation for β-MBA in $\tilde{O}(nm/\epsilon)$ time.*

Proof: Armed with the Property 2.2.9 which implies $S_i \geq L(\alpha)B_i$ for all i, the proof of Theorem 2.2.6 can be modified (the only difference is we need to run till $\alpha_i > 1 - 1/(4-\beta)m$ instead of $(1 - 1/3m)$), to show that the algorithm terminates with an ϵ-valid allocation with all agents paid for. The proof of the factor follows from the proof of Theorem 2.2.5 and Claim 2.2.8. □

2.3 Inapproximability of MBA and related problems

In this section we study the inapproximability of MBA and the related problems as stated in the introduction. The main theorem of this section is the following 15/16 hardness of approximation factor for MBA.

Theorem 2.3.1 *For any $\epsilon > 0$, it is NP-hard to approximate* MBA *to a factor $15/16 + \epsilon$. This holds even for* uniform *instances.*

We give a reduction from MAX-3-LIN(2) to MBA to prove the above theorem. The MAX-3-LIN(2) problem is as follows: Given a set of m equations in n variables over $GF(2)$, where each equation contains exactly 3 variables, find an assignment to the variables to maximize the number of satisfied equations. Håstad, in his seminal work [29], gave the following theorem.

Theorem 2.3.2 *[29] Given an instance I of* MAX-3-LIN(2)*, for any $\delta, \eta > 0$, its NP hard to distinguish between the two cases:* YES: *There is an assignment satisfying $(1-\delta)$-fraction of equations, and* NO: *No assignment satisfies more than $(1/2+\eta)$-fraction of equations.*

Let I be an instance of MAX-3-LIN(2). Denote the variables as $x_1, \cdots, x_n$. Also let $deg(x_i)$ be the *degree* of variable x_i i.e. the number of equations in which variable x_i occurs. Note that $\sum_i deg(x_i) = 3m$. We construct an instance $R(I)$ of MBA as follows:

- For every variable x_i, we have two agents which we label as $\langle x_i : 0 \rangle$ and $\langle x_i : 1 \rangle$, corresponding to the two assignments. The budget of both these agents is $4deg(x_i)$ (4 per equation).
- There are two kinds of items. For every variable x_i, we have a *switch item* s_i. Both agents, $\langle x_i : 0 \rangle$ and $\langle x_i : 1 \rangle$, bid their budget $4deg(x_i)$ on s_i. No one else bids on s_i.

- For every equation $e : x_i + x_j + x_k = \alpha$ ($\alpha \in \{0, 1\}$), we have 4 kinds of items corresponding to the four assignments to x_i, x_j, x_k which satisfy the equation: $\langle x_i : \alpha, x_j : \alpha, x_k : \alpha \rangle$, $\langle x_i : \alpha, x_j : \bar{\alpha}, x_k : \bar{\alpha} \rangle$, $\langle x_i : \bar{\alpha}, x_j : \bar{\alpha}, x_k : \alpha \rangle$ and $\langle x_i : \bar{\alpha}, x_j : \alpha, x_k : \bar{\alpha} \rangle$. For each equation, we have 3 copies of each of the four items. The set of all 12 items are called *equation items*, and denoted by S_e. Thus we have $12m$ equation items, in all.

  For every equation item of the form $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k \rangle$, only three agents bid on it: the agents $\langle x_i : \alpha_i \rangle$, $\langle x_j : \alpha_j \rangle$ and $\langle x_k : \alpha_k \rangle$. The bids are of value 1 each.

The following lemma is not hard to see.

Lemma 2.3.3 *There always exists an optimal solution to $R(I)$ in which every switch item is allocated.*

Proof: Suppose there is a solution which is not valid. Thus there is a switch item s_i which is not allocated. Allocating s_i to either $\langle x_i : 0 \rangle$ or $\langle x_i : 1 \rangle$ and de-allocating the items allocated to the agent can only increase the value of the allocation. □

Since agents who get switch items exhaust their budget, any more equation items given to them generate no extra revenue. We say that an equation item can be allocated in $R(I)$ only if it generates revenue, that is, it is not allocated to an agent who has spent all his budget.

Lemma 2.3.4 *Given an assignment of variables by $R(I)$, if an equation e is satisfied then all the 12 items of S_e can be allocated in $R(I)$. Otherwise, at most 9 items of S_e can be allocated in $R(I)$.*

Proof: If an equation e is satisfied, then there must be one equation item $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k \rangle$ such that x_r is assigned α_r ($r = i, j, k$) in the assignment by $R(I)$ (that is the switch item s_r is given to $\langle x_r : \bar{\alpha_r} \rangle$). Assign the 12 items of S_e as follows: give one

the three copies of $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k \rangle$ to agents $\langle x_i : \alpha_i \rangle$, $\langle x_j : \alpha_j \rangle$ and $\langle x_k : \alpha_k \rangle$. Note that none of them have got the switch item. Moreover, for the other items in S_e, give all 3 copies of $\langle x_i : \alpha_i, x_j : \bar{\alpha_j}, x_k : \bar{\alpha_k} \rangle$ to agent $\langle x_i : \alpha_i \rangle$, and similarly for the three copies of $\langle x_i : \bar{\alpha_i}, x_j : \alpha_j, x_k : \bar{\alpha_k} \rangle$ and $\langle x_i : \bar{\alpha_i}, x_j : \bar{\alpha_j}, x_k : \alpha_k \rangle$. Since each agent gets 4 items, he does not exhaust his budget.

If an equation e is *not* satisfied, then observe that there must be an equation item $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k \rangle$ such that x_r is assigned $\bar{\alpha_r}$ $(r = i, j, k)$ in the assignment. That is, all the three agents bidding on this item have their budgets filled up via switch items. Thus none of the copies of this equation item can be allocated, implying at most 9 items can be allocated. □

The following two lemma along with Håstad's theorem prove the hardness for maximum budgeted allocation given in Theorem 2.3.1.

Lemma 2.3.5 *If $OPT(I) \geq m(1-\epsilon)$, then the maximum budgeted allocation revenue of $R(I)$ is at least $24m - 12m\epsilon$.*

Proof: Allocate the switch elements in $R(I)$ so that the *assignment* of variables by $R(I)$ is same as the *assignment* of I. That is, if x_i is assigned 1 in the solution to I, allocate s_i to $\langle x_i : 0 \rangle$, and vice versa if x_i is assigned 0. For every equation which is satisfied, allocate the 12 equation items as described in Lemma(2.3.4). Since each agent gets at most 4 items per equation, it gets at most $4deg(x_i)$ revenue which is under his budget. Thus the total budgeted allocation gives revenue: gain from switch items + gain from equation items $= \sum_i 4deg(x_i) + 12m(1-\epsilon) = 24m - 12m\epsilon$. □

Lemma 2.3.6 *If $OPT(I) \leq m(1/2 + \eta)$, then the maximum budgeted allocation revenue of $R(I)$ is at most $22.5m + 3m\eta$*

Proof: Suppose not. i.e . the maximum revenue of $R(I)$ is strictly greater than $22.5m + 3m\eta$. Since the switch items can attain at most $12m$ revenue, $10.5m + 3m\eta$

must have been obtained from equation items. We claim that there must be strictly more than $m(1/2+\eta)$ equations so that at least 10 out of their 12 equation items are allocated. Otherwise the revenue generated will be at most $12m(1/2+\eta)+9m(1/2-\eta) = 10.5m + 3m\eta$ The contradiction follows from Lemma(2.3.4). □

2.3.1 Hardness of SMW with demand oracle

As noted in Section 2.0.3, MBA is a special case of SMW. Thus the hardness of approximation in Theorem 2.3.1 would imply a hardness of approximation for SMW with the demand oracle, if the demand oracle could be simulated in poly-time in the hard instances of MBA. Lemma 2.3.8 below shows that this indeed is the case which gives the following theorem.

Theorem 2.3.7 *For any $\epsilon > 0$, it is NP-hard to approximate submodular welfare with demand queries to a factor $15/16 + \epsilon$.*

Lemma 2.3.8 *Given any instance I of* Max-3-Lin(2)*, in the corresponding instance $R(I)$ as defined in Section 2.3 the demand oracle can be simulated in polynomial time.*

Proof: We need to show that for any agent i and given prices $p_1, p_2 \cdots$ to the various items, one can find a subset of items S which maximizes $(\min(B_i, \sum_{j\in S} b_{ij}) - \sum_{j\in S} p_j)$. Call such a bundle the optimal bundle. Observe that in the instance $R(I)$, the bid of an agent i is 1 on an equation item and B_i on the switch item. Therefore, the optimal bundle S either consists of just the switch item or consists of B_i equation items. The best equation items are obviously those of the smallest price and thus can be found easily (in particular in polynomial time). □

2.3.2 Hardness of β-MBA

The hardness reduction given above can be easily modified to give a hardness result for β-MBA, for any constant $1 \geq \beta > 0$. Note that the budget of an agent is four

times the degree of the analogous variable. We increase the budget of agents $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$ to $\frac{1}{\beta}4deg(x_i)$. For each agent, introduce dummy items so that the total bid of an agent on these dummy items is $(\frac{1}{\beta} - 1)$ times its original budget. The rest of the reduction remains the same. Call this new instance β-$R(I)$.

Claim 2.3.9 *We can assume that in any optimal allocation, all the dummy items are assigned*

Proof: If a dummy item is not assigned and assigning it exceeds the budget of the agent implies the agent must be allocated an equation item. De-allocating the equation item and allocating the dummy item gives an allocation of at least the original cost. □

Once the dummy items are assigned, the instance reduces to the original instance. We have the following analogous lemmas of Lemma2.3.5 and Lemma2.3.6.

Lemma 2.3.10 *If $OPT(I) \geq m(1-\epsilon)$, then the maximum budgeted allocation revenue of β-$R(I)$ is at least $24m-12m\epsilon+24m(\frac{1}{\beta}-1)$. If $OPT(I) \leq m(1/2+\eta)$, then the maximum budgeted allocation revenue of $R(I)$ is at most $22.5m + 3m\eta + 24m(\frac{1}{\beta} - 1)$*

Proof: The extra $24m(\frac{1}{\beta} - 1)$ is just the total value of the dummy items which is obtained in both cases. □

The above theorem with Håstad's theorem gives the following hardness result for β-MBA.

Theorem 2.3.11 *For any $\epsilon > 0$, it is NP-hard to approximate β-MBA to a factor $1 - \beta/16 + \epsilon$.*

2.3.3 Hardness of GAP

To remind, in the generalized assignment problem (GAP) we have n bins each with a capacity B_i. There are a set of items with item j having a profit p_{ij} and size s_{ij}

corresponding to bin i. The objective is to find an allocation of items to bins so that no capacities are violated and the total profit obtained is maximized.

One of the bottlenecks for getting a better lower bound for MBA is the extra contribution of switch items which are always allocated irrespective of I. A way of decreasing the effect of these switch items is to decrease their value. In the case of MBA this implies reducing the bids of agents on switch items. Note that this might lead to an agent having a switch item and an equation item as he has budget remaining, and thus the allocation does not correspond to an assignment for the variables. This is where the generality of GAP helps us: the switch item will have a reduced profit but the size will still be the capacity of the agent (bin). However, since we would want switch items to be *always* allocated, we cannot reduce their profits by too much. We use this idea to get the following theorem.

Theorem 2.3.12 *For any $\epsilon > 0$, it is NP-hard to approximate GAP to a factor $10/11 + \epsilon$.*

We now describe our gadget more in detail. The gadget is very much like the one used for MBA.

- For every variable x_i, we have two bins $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$, corresponding to the two assignments. The capacity of both these bins is $2deg(x_i)$ (2 per equation).

- There are two kinds of items. For every variable x_i, we have a *switch item* s_i. s_i can go to only one of the two bins, $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$. Its capacity for both bins is $2deg(x_i)$ while its profit is $deg(x_i)/2$.

- For every equation of the form $e : x_i+x_j+x_k = \alpha$ ($\alpha \in \{0, 1\}$), we have a set S_e of 4 items, called *equation items*, corresponding to the four assignments to x_i, x_j, x_k which satisfy the equation: $\langle x_i : \alpha, x_j : \alpha, x_k : \alpha\rangle$, $\langle x_i : \alpha, x_j : \bar{\alpha}, x_k : \bar{\alpha}\rangle$, $\langle x_i : \bar{\alpha}, x_j : \bar{\alpha}, x_k : \alpha\rangle$ and $\langle x_i : \bar{\alpha}, x_j : \alpha, x_k : \bar{\alpha}\rangle$. Thus we have $4m$ equation

items, in all. Every equation item of the form $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k \rangle$, can go to any one of the three bins $\langle x_i : \alpha_i \rangle$, $\langle x_j : \alpha_j \rangle$ and $\langle x_k : \alpha_k \rangle$. The profit and size for each of this bins is 1.

We will use $R(I)$ to refer to the instance of GAP obtained from the instance I of MAX-3-LIN(2). Lets say a solution to an instance $R(I)$ of GAP is a *k-assignment* solution if exactly k switch items have been allocated. We will use *valid-assignment* to refer to the *n-assignment* solution.

Lemma 2.3.13 *For every solution of $R(I)$ which is a k-assignment solution such that variable x_i is unassigned (i.e. item s_i is neither allocated to $\langle x_i : 0 \rangle$ nor $\langle x_i : 1 \rangle$) there exists a k-assignment solution of at least the same value in which x_i is unassigned and both $\langle x_i : 0 \rangle$ and $\langle x_i : 1 \rangle$ together gets atmost one item from S_e for every equation e of x_i. i.e. they both get a total of atmost $deg(x_i)$ items.*

Proof: Suppose not. i.e. there exists an equation e (say $x_i + x_j + x_k = \alpha$) s.t. both $\langle x_i : 0 \rangle$ and $\langle x_i : 1 \rangle$ together gets atleast two items out of S_e. Notice that the switch items of x_j and x_k can fill the capacity of atmost one bin out of their respective two bins. Suppose the free bins are $\langle x_j : \alpha \rangle$ and $\langle x_k : \alpha \rangle$ (The other cases can be considered similarly). Now except for the item $\langle x_i : \alpha, x_j : \bar{\alpha}, x_k : \bar{\alpha} \rangle$, all the other 3 items in S_e are wanted by $\langle x_j : \alpha \rangle$ and $\langle x_k : \alpha \rangle$. By the above property, all of these 3 items can be allcocated to the bins $\langle x_j : \alpha \rangle$ and $\langle x_k : \alpha \rangle$. Thus we can reallocate the items of S_e such that atmost one item out of S_e is allocated to the corresponding bins of variable x_i without decreasing the profit. □

Now by the above lemma, for any unassigned variable x_i in a *k-assignment* solution, one of the bins out of $\langle x_i : 0 \rangle$ and $\langle x_i : 1 \rangle$ will have atmost $deg(x_i)/2$ items. We can remove these items and allocate the switch element of x_i without reducing the profit. Thus we get the following corollary.

Corollary 2.3.14 *For every optimal solution of $R(I)$ which is a k-assignment solution there exists a (k+1)-assignment solution which is also optimal. Therefore, there exists a optimal solution of $R(I)$ which is a valid assignment*

Now using arguments similar to lemma 2.3.4 , one can show the following:

Lemma 2.3.15 *Consider a valid-assignment solution (say v-sol) of R(I). If an equation e is satisfied by the assignment of variables given by v-sol then all the 4 items of S_e can be allocated in v-sol. Otherwise atmost 3 items out of S_e can be allocated in v-sol.*

Now using arguments similar to lemma 2.3.5 and 2.3.6, one can prove the following lemma which along with Håstad's theorem implies Theorem 2.3.12.

Lemma 2.3.16 *Let I be an instance of* Max-3-Lin(2) *and R(I) be its reduction to GAP, then:*

- *If $OPT(I) \geq m(1-\epsilon)$, then the maximum profit of $R(I)$ is at least $5.5m - 4m\epsilon$.*
- *If $OPT(I) \leq m(1/2+\eta)$, then the maximum profit of $R(I)$ is at most $5m + m\eta$*

Proof: Suppose $OPT(I) \geq m(1-\epsilon)$. Allocate switch elements in $R(I)$ so that the assignment of variables is same as the one given by optimal solution of I. Now the profit from switch items equals: $\sum_i (deg(x_i)/2) = 3m/2$. Also by lemma 2.3.15, the profit from *equation* items is atleast $4m(1-\epsilon)$. Combining both, we get the first part of the lemma.

Suppose $OPT(I) \leq m(1/2+\eta)$. By corollary 2.3.14, there exists a optimum solution of $R(I)$ which is a *valid* assignment. Consider any such solution. Now the claim is that for no more than $m(1/2+\eta)$ equations, all the 4 items of S_e's can be allocated in this solution. If they do, then along with lemma 2.3.15 it contradicts the fact that $OPT(I) \leq m(1/2+\eta)$. Thus profit from equation items can be atmost:

$4m(1/2+\eta)+3m(1/2-\eta) = 7m/2+m\eta$. Hence total profit can be atmost $3m/2+7m/2+m\eta$. $\square$

2.3.4 Hardness of weighted MSSF

Given an undirected graph G, the unweighted maximum spanning star forest problem (MSSF) is to find a forest with as many edges such that each tree in the forest is a star. The edge-weighted MSSF (eMSSF) is the natural generalization with weights on edges. The node-weighted MSSF (nMSSF) has weights on vertices and the weight of a star is the weight on the leaves. If the star is just an edge, then the weight of the star is the maximum of the weights of the end points.

It is clear that the non-trivial part of the above problem is to identify the vertices which are the centers of the stars. Once more, we reduce Max-3-Lin(2) to both eMSSF and nMSSF. Let us discuss the edge-weighted MSSF first. For every variable x in an instance of Max-3-Lin(2), we introduce two vertices: $\langle x:0\rangle$, $\langle x:1\rangle$. The interpretation is clear: we will enforce that exactly one of these vertices will be the center which will coincide with the assignment in the Max-3-Lin(2) instance. Such an enforcing is brought about by putting a heavy cost edge between the vertices. For every equation we will add vertices as we did in the reduction to GAP.

In the node-weighted MSSF, we need to add an extra vertex, the *switch vertex*, along with the two vertices $\langle x:0\rangle$, $\langle x:1\rangle$. These vertices form a triangle and have a weight high enough to ensure that exactly one of $\langle x:0\rangle$, $\langle x:1\rangle$ is chosen as a center in any optimum solution to nMSSF.

We remark that Chen et.al [15] also use a similar gadget as the one above, although their reduction is from a variation of MAX-3-SAT and thus their results are weaker.

Hardness of eMSSF: Let I be an instance of Max-3-Lin(2). Denote the variables

as $x_1, \cdots, x_n$. Also let $deg(x_i)$ be the *degree* of variable x_i i.e. the number of equations in which variable x_i occurs. Note that $\sum_i deg(x_i) = 3m$. We construct an instance $E(I)$ of eMSSF as follows:

- For every variable x_i, we have two variables which we label as $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$, corresponding to the two assignments. These variables are called variable vertices. There is an edge between them of weight $deg(x_i)/2$.
- For every equation $e : x_i + x_j + x_k = \alpha$ ($\alpha \in \{0, 1\}$), we have 4 vertices corresponding to the four assignments to x_i, x_j, x_k which satisfy the equation: $\langle x_i : \alpha, x_j : \alpha, x_k : \alpha\rangle$, $\langle x_i : \alpha, x_j : \bar{\alpha}, x_k : \bar{\alpha}\rangle$, $\langle x_i : \bar{\alpha}, x_j : \bar{\alpha}, x_k : \alpha\rangle$ and $\langle x_i : \bar{\alpha}, x_j : \alpha, x_k : \bar{\alpha}\rangle$. This set of vertices are called equation vertices denoted by S_e. Thus we have $4m$ equation vertices in all. Each equation vertex of the form $\langle x_i : \alpha_i, x_j : \alpha_j, x_k : \alpha_k\rangle$ is connected to three variable vertices: $\langle x_i : \alpha_i\rangle$, $\langle x_j : \alpha_j\rangle$ and $\langle x_k : \alpha_k\rangle$. The weight of all these edges is 1. Thus, the degree of every equation vertex is 3 and the degree of every variable vertex $\langle x_i : 0\rangle$ or $\langle x_i : 1\rangle$ is $2deg(x_i)$.

Lemma 2.3.17 *Given any solution to $E(I)$, there exists a solution of at least the same weight where the centers are exactly one variable vertex per variable.*

Proof: Firstly note that if none of the variable vertices are centers then one can make one of them a center and connect the other to it and get a solution of higher cost (note that the degrees of the variables in the equations can be assumed to be bigger than 4 by replication). The proof is in two steps. Call a variable $x_i \in I$ *unassigned* if both of $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$ is a center. Call a solution of $E(I)$ k-satisfied if exactly k of the variables are assigned. The claim is that there exists a solution of equal or more weight which is n-satisfied. We do this via induction.

We show that if a solution to $E(I)$ is k-satisfied with $k < n$,then we can get a solution of at least this weight which is $k + 1$-satisfied. Pick a variable x_i which is

unassigned. For every equation $e : x_i + x_j + x_k = 0$, say, containing x_i we claim that one can assume of the four equation vertices in S_e, only one is connected to $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$. This is because at least one of the two variable vertices corresponding to both x_j and x_k are centers. Suppose these are $\langle x_j : 0\rangle$ and $\langle x_k : 0\rangle$. Now note that of the four vertices in S_e only $\langle x_i : 0, x_j : 1, x_k : 1\rangle$ is not neighboring to either of these centers. The three remaining can be moved to these without any decrease in the weight of the solution and the claim follows.

Thus, we can assume that for every unassigned variable x_i in the k-satisfied solution, one of the two variable vertices $\langle x_i : 0\rangle$ or $\langle x_i : 1\rangle$ (say $\langle x_i : 0\rangle$), is connected to at most $deg(x_i)/2$ equation vertices. Therefore, disconnecting all the equation items connected to $\langle x_i : 0\rangle$, making it a leaf and connecting it to $\langle x_i : 1\rangle$, gives a $k+1$-satisfied solution of weight at least the original weight. □

Now we get the hardness of eMSSF using the theorem of Håstad.

Theorem 2.3.18 *For any $\epsilon > 0$, it is NP-hard to approximate edge-weighted MSSF to a factor $10/11 + \epsilon$.*

Proof: The proof follows from the following two calculations and Theorem 2.3.2.

- If $OPT(I) \geq m(1-\delta)$, then the maximum profit of $E(I)$ is at least $5.5m - 4m\delta$. For every variable x_i, if the assignment of x_i is $\alpha \in \{0, 1\}$, make $\langle x_i : \alpha\rangle$ the center. Observe that for every satisfied equation $e : x_i + x_j + x_k = \alpha$, all the four vertices of S_e can be connected to a center. Thus the weight of $E(I)$ is at least $\sum_i deg(x_i)/2 + 4m(1-\delta) = 5.5m - 4m\delta$.

- If $OPT(I) \leq m(1/2+\eta)$, then the maximum profit of $E(I)$ is at most $5m + m\eta$ From the claim above we can assume for each variable x_i, one of its two variable vertices is a center. This defines an assignment of truth values to the variables and around half of the equations are not satisfied by this assignment. The observation is that for any unsatisfied equation $e : x_i + x_j + x_k = \alpha$, one of

the four equation vertices in S_e is not connected to any center. Thus, the total weight of any solution is at most $\sum_i deg(x_i)/2 + 4m(1/2+\eta) + 3m(1/2-\eta) = 5m + m\eta$.

□

Hardness of nMSSF: Let I be an instance of Max-3-Lin(2). The only difference between the instance of nMSSF, $N(I)$, and the eMSSF $E(I)$ is that for every variable $x_i \in I$, along with the variable vertices $\langle x_i : 0\rangle$ and $\langle x_i : 1\rangle$, we have a switch vertex s_i. The three vertices from a triangle and the node-weights of all of them are $deg(x_i)/2$. The rest of the instance of $N(I)$ is exactly like $E(I)$ with the edge-weights being replaced by node-weights of 1 on the equation vertices.

The reason we require the third switch vertex per variable is that otherwise we cannot argue that in any solution to $N(I)$ at least one of the variable vertices should be a center. With the switch vertex, we can argue that this is the case. If none of the variable vertices is a center, then the switch item is not connected to any vertex. Thus making any one of the variable vertices as a center connected to the switch item gives a solution to $N(I)$ of weight at least the original weight.

Lemma 2.3.17 now holds as in the case of $E(I)$ and thus similar to Theorem 2.3.19 we have the following hardness of node-weighted MSSF.

Theorem 2.3.19 *For any $\epsilon > 0$, it is NP-hard to approximate edge-weighted MSSF to a factor $13/14 + \epsilon$.*

Proof: The proof follows from the following two calculations and Theorem 2.3.2.

- If $OPT(I) \geq m(1-\delta)$, then the maximum profit of $N(I)$ is at least $7m - 4m\delta$. For every variable x_i, if the assignment of x_i is $\alpha \in \{0,1\}$, make $\langle x_i : \alpha\rangle$ the center. Connect the switch item and the vertex $\langle x_i : \bar{\alpha}\rangle$ to this center. Observe that for every satisfied equation $e : x_i + x_j + x_k = \alpha$, all the four

vertices of S_e can be connected to a center. Thus the weight of $N(I)$ is at least $\sum_i deg(x_i) + 4m(1-\delta) = 7m - 4m\delta$.

- If $OPT(I) \leq m(1/2+\eta)$, then the maximum profit of $N(I)$ is at most $6.5m+m\eta$ From the claim above we can assume for each variable x_i, one of its two variable vertices is a center. This defines an assignment of truth values to the variables and around half of the equations are not satisfied by this assignment. The observation is that for any unsatisfied equation $e : x_i + x_j + x_k = \alpha$, one of the four equation vertices in S_e is not connected to any center. Thus, the total weight of any solution is at most $\sum_i deg(x_i) + 4m(1/2+\eta) + 3m(1/2-\eta) = 6.5m + m\eta$.

□

2.4 Discussion

In this paper we studied the maximum budgeted allocation problem and showed that the true approximability lies between 3/4 and 15/16. Our algorithms were based on a natural LP relaxation of the problem and we, in some sense, got the best out if it: the integrality gap of the LP is 3/4. An approach to get better approximation algorithms might be looking at stronger LP relaxations to the problem. One such relaxation is the *configurational LP relaxation* which we describe in detail in section 2.4.1. Configurational LPs have been used for other allocation problems; in fact the best known approximation algorithm of Feige and Vondrak [20] for SMW and GAP proceeds by rounding the solution of the LP. We demonstrate an example which shows that the integrality gap of this LP is at most 5/6. We leave the pinning down of the integrality gap as an open question and we believe that the resolution might require some newer techniques in addition to the ones developed in this paper.

Another direction of research is improving the hardness of approximation factor. We define a natural extention of SAT below which seems to be at the core of MBA and other allocation problems like GAP and SWM.

Definition 3 BUDGETED-3SAT(b)*: Given a formula* $\phi = C_1 \wedge C_2 \cdots \wedge C_m$*, where each clause* C_i *is a disjunction of three literals, let the degree,* $deg(l)$*, of a literal* l *be the number of clauses the literal appears in. The* BUDGETED-3SAT(b) *asks for an assignment of variables and a maximum sized pairing* $\{l_i, C_i\}$ *for every clause* C_i *and* $l_i \in C_i$ *such that the assignment of the literal satisfies the disjunction* C_i*, and every literal* l *appears in at most* $b \cdot deg(l)$ *pairings.*

Clearly, MAX-3-SAT is just the BUDGETED-3SAT(1) problem. Our hardness result essentially shows that BUDGETED-3SAT(b) (for any budget b) can be reduced to MBA, and a hardness of $\rho(b)$ for BUDGETED-3SAT(b) implies a hardness of $\frac{2\rho(b)+3b}{2+3b}$. Implicit in our reduction is the fact that BUDGETED-3SAT(2/3) is 7/8-hard which gives us a 15/16 hardness for MBA.

For general values of b however the approximibility BUDGETED-3SAT(b) is open. In fact, for general b, it is not even clear if BUDGETED-3SAT(b) is easier or harder than MAX-3-SAT. Obtaining hardness results for BUDGETED-3SAT(b) seems to be an interesting approach towards the inapproximibility of allocation problems.

2.4.1 The configurational LP relaxation for MBA

In this relaxation, we have a variable $x_{i,S}$ for every agent i and every subset of items $S \subseteq Q$. The LP is as follows

$$\begin{aligned} \text{Max} \quad & \{\sum_i \sum_{S \subseteq Q} u_i(S) x_{i,S} \\ \text{s.t.:} \quad & \forall i \in A, \quad \sum_{S \subseteq Q} x_{i,S} \leq 1; \\ & \forall j \in Q, \sum_{i, S \subseteq Q : j \in S} x_{i,S} \leq 1; \\ & \forall i \in A, S \subseteq Q, \qquad x_{i,S} \geq 0\} \end{aligned} \tag{4}$$

The first constraint implies that each agent gets at most one subset of items. The second implies that each item is at most one subset. The value generated on giving a subset S to agent i is $u_i(S) = \min(B_i, \sum_{j \in S} b_{ij})$.

Solving the LP: Observe that the LP has exponentially (in n and m) many variables and an obvious question is how to solve the LP. One does so by going to the dual LP which will have exponentially many constraints but only polynomially many variables. Such a trick is now standard first used by Carr and Vempala [11]. The dual of LP(4) is as follows

$$\begin{aligned} \text{Min} \quad & \{\sum_{i \in A} \alpha_i + \sum_{j \in Q} p_j \\ \text{s.t.:} & \forall i \in A, S \subseteq Q, \quad \alpha_i + \sum_{j \in S} p_j \geq u_i(S); \\ & \forall i \in A, \forall j \in Q, \quad \alpha_i, p_j \geq 0\} \end{aligned} \tag{5}$$

Suppose, for the time being, the LP(5) has a separation oracle: Given (α_i, p_j) one can say in polynomial time if it is feasible for LP(5) or find a subset of agents S with $\alpha_i + \sum_{j \in S} p_j < u_i(S)$. If so, then the ellipsoid algorithm can be used to solve the LP by making a polynomial number of queries to the separation oracle. Moreover, the

subsets returned by the separation oracle are enough to describe the optimal solution to the dual. In other words, in the primal LP(4), only the variables corresponding to these constraints need be positive and the rest can be set to 0 and the optimum is not changed. Since they are only polynomially many, LP(4) can now be solved in polynomial time. Moreover, if one had an r-separation oracle ($r \leq 1$): Given (α_i, p_j) one can say in polynomial time if $\frac{1}{r} \cdot (\alpha_i, p_j)$ is feasible for LP(5) or find a subset of agents S with $\alpha_i + \sum_{j \in S} p_j < u_i(S)$; then the above argument can be used to get an r-approximation for the primal LP(4).

Note that the separation problem for LP(5) is precisely the demand oracle problem described Section 2.0.3: given prices p_j to items, for every agent i find a subset of items maximizing $(u_i(S) - \sum_{j \in S} p_j)$ where $u_i(S) = \min(B_i, \sum_{j \in S} b_{ij})$. The problem is NP-hard with a reduction from PARTITION. Given an instance of partition of n integers $b_1, b_2, \cdots, b_n$ and a target integer B, consider the instance of the separation problem with n items having bids b_1 to b_n and prices $b_1/2, \cdots, b_n/2$ and budget B. Note that the maximum value of the separation problem is at most $B/2$ and moreover the optimum is $B/2$ if and only if there is a subset of the integers adding exactly to B.

We now demonstrate an $(1 - \epsilon)$-separation oracle for LP(5) using the FPTAS for the knapsack problem. The sketch below is not the fastest implementation as we interested mainly in the existence of polynomial time algorithms. In the knapsack problem, we are given a capacity of B and n items having profits p_j and weight w_j and the goal is to obtain a subset of items having maximum profit with the total weight being less than B. The problem in NP-hard, but an FPTAS exists. Moreover, there is an exact algorithm which runs in time $O(n^2 P)$ where P is the largest profit. Given the separation problem for LP(5), we consider the items in any arbitrary order. The first item has a bid of b_1 and price p_1. Suppose the item is picked in the optimum solution, call it S. If so, then $\sum_{j \in S} b_{ij} \leq B + b_1$, as otherwise one could discard the

first item and get a better solution. Thus the optimum solution of the separation problem given the first item is picked is precisely

$$\max_{0 \leq x \leq b_1} \{\text{Knapsack}[\{(b_2 - p_2, b_2), \cdots, (b_n - p_n, b_n)\}, B - x] + (x - p_1)\}$$

$(x - p_1)$ is the value of the item 1 after the items from $b_2, \cdots, b_n$ use up $B - x$ of the budget. One can repeat the procedure n times removing one item at a time and in the end taking the best solution over all iterations. This gives the optimum solution in time $O(n^3 B^2)$, where B is the budget.

To make the above algorithm run in polynomial time, we round down to the nearest integer all the budgets, bids and prices by a factor of $\epsilon B/n$. The solution to this reduced instance can be found in time $O(n^5/\epsilon)$ and the same solution, scaled back, can be shown to be within $(1-\epsilon)$ of the optimal solution (see for example, [63]).

Integrality gap of the configurational LP: In the next theorem we show that the integrality gap of the configurational LP is between 3/4 and 5/6. The lower bound follows basically by showing that the value of LP(4) is at most the value of LP(1) (and thus is a better upper bound on the optimum). The upper bound follows from an example which we demonstrate below.

Theorem 2.4.1 *The integrality gap of the configurational LP of MBA is between* 3/4 *and* 5/6.

Proof: An easy way to see that the configurational LP is stronger than LP(1) is by looking at the duals of both LP's. One can show that any solution (α_i, p_j) to LP(2) corresponds to a feasible solution $(B_i \alpha_i, p_j)$ to LP(5) of equal value. Thus the configurational LP value is smaller than that of LP(1).

The 5/6-example is as follows: The instance consists of 4 agents a_1, b_1, a_2, b_2. a_1, a_2 have a budget of 1, b_1, b_2 have a budget of 2. There are five items: c, x_1, y_1 and

x_2, y_2. Only b_1 and b_2 bid on c and bid 2. For $i = 1, 2$, a_i and b_i each bid on x_i and y_i, and the bid is 1. Once again, if c is given to b_1, then either a_2 or b_2 ends up spending 1 less than his budget. Thus, the optimum MBA solution is 5. But there is a solution to the configurational LP(4) of value 6. The sets are $S_1 = \{x_1\}$, $S_2 = \{y_1\}$, $S_3 = \{x_2\}$, $S_4 = \{y_2\}$, $S_5 = \{c\}$,$S_6 = \{x_1, y_1\}$ and $S_7 = \{x_2, y_2\}$. The solution is: $x_{a_1,S_1} = x_{a_1,S_2} = x_{a_2,S_3} = x_{a_2,S_4} = 1/2$ and $x_{b_1,S_6} = x_{b_1,S_5} = x_{b_2,S_5} = x_{b_2,S_7} = 1/2$. □

CHAPTER III

BUDGETED ALLOCATION: RANDOM ARRIVAL MODEL

In this chapter we study online budgeted allocation problem in which there is a *random* stream of item (queries) arriving online, and each item, as it arrives, has to be allocated to one of several bidders who are bidding different amounts for it. Each bidder also has a total budget limit. The goal is to maximize the total value of the allocation. As the online setting is mostly motivated by the sponsored search auctions, we will assume that the bid to budget ratio is small.

The algorithms used by the search companies currently are, in essence, *Greedy* : Select the highest bidders for each query[1]. It can be shown that the Greedy algorithm has a competitive ratio of 1/2 in the standard online worst-case competitive analysis model. In [48] a deterministic algorithm with a competitive ratio of $1 - 1/e \simeq 0.63$ was presented, which was proved to be optimal, even over randomized algorithms[2].

In spite of the elegance and optimal competitive ratio of this algorithm, there remains some dissatisfaction with the use of worst case analysis in that we are unlikely to see a worst-case sequence of queries as input. In this paper we make a more distributional assumption about the query sequence: the queries arrive in a random permutation, i.e. the *set* of queries is still arbitrary, but the *order* of queries is random. This leads us to formulate the following combinatorial problem :

[1]The bids are normalized by other factors, such as click-through-rate and relevance (see Google, Yahoo!, MSN websites). Furthermore each winner is charged the next highest bid. For simplicity of presentation, we will assume that the bids are already normalized, and furthermore, that we will display only one ad per query, and hence need to determine a single winner per query. We do not consider the issue of second price in our setting, and assume that a winning bidder pays his own bid.

[2][48] made a natural modeling assumption that all the bids of a bidder are very small compared to his budget. We will also make this assumption.

There are m bidders. Bidder i has a budget of B_i. There is a (unknown) set Q of n queries. For $q \in Q$, and $i \in [m]$, bid_{iq} is the bid of bidder i for query q. The queries arrive online one at a time, in a random permutation of Q. The algorithm has to allocate each query, as it arrives, to a bidder. Let Q_i be the set of queries allocated to bidder i at the end of the input sequence. The revenue of the algorithm is defined as $\sum_{i=1}^{m} \min\{B_i, \sum_{q \in Q_i} bid_{iq}\}$. The goal of the algorithm is to maximize its revenue.

We also study this allocation problem in the more standard *i.i.d.* input model, in which there is an unknown distribution on queries, and the next query in the sequence is picked independently from this distribution.

Understanding the budgeted allocation problem in distributional models is important, and was an open question in [48] and in [43] (see also the approach in [47]). The random permutations model has been studied, e.g., in the classic secretary problems [19], recent work in auctions [4] and in data streams [28] with the motivation that an adversary may have control over the set of inputs, but not have control over the order of the input, which may follow some random process.

In the problem above, we will assume that for each bidder $i \in m$: $\max_{q \in Q} bid_{iq}$ is very small compared to B_i. This is a reasonable assumption for the sponsored search auction model, and was also made in [48]. But in fact, our algorithm and analysis work for a larger class, encompassing this assumption. We define this class formally in Section 3.2, but mention here that this class also contains the problems of bipartite matching (all bids 0 or 1, all budgets 1) and b-matching, for any positive integer b (all bids 0 or 1, all budgets b). The *offline* version of this problem was studied in chapter 2 for the case when bids are not restricted to be small compared to budgets (when the bids are very small compared to budgets then LP rounding gives a factor very close to 1). Finally, we note that the problem defined above is a general combinatorial

allocation problem, closely related to classic matching problems, and with possible applications beyond the current motivation.

3.0.2 Main Results

Our main result is an analysis to show that Greedy has competitive ratio $1 - 1/e$ in the random permutation input model, as well as in the i.i.d. model. This is to be compared with the ratios in the worst case input model: 1/2 for Greedy and the optimal $1 - 1/e$ for the algorithm in [48]. We also show that our analysis is tight by providing examples in the two models for which Greedy has ratio exactly $1 - 1/e$. We also show a lower bound that no deterministic algorithm can have a ratio better than 3/4 and no randomized algorithm can have a ratio better than 5/6 for the case of matching in the random permutation model.

Along the way, we provide a direct proof for the RANKING algorithm of [37] for the online bipartite matching problem (see below for details).

3.0.3 Techniques:

From a technical perspective our work can be seen as simplifying and generalizing the classic work of Karp, Vazirani and Vazirani on online bipartite matching (matching boys to girls) in the worst case input model [37]. Their RANKING algorithm fixes beforehand a random permutation of the boys, and matches each arriving girl to the highest available neighbor in the permutation. They show a duality result that the performance of RANKING in the worst case is the same as the performance of Greedy in the random permutations model (boys arrive in a random permutation). They show that this competitive ratio is $1 - 1/e$, by analyzing a "virtual algorithm" called EARLY, which is allowed to refuse to match a boy if the "correct" girl was already matched off.

This proof, however, seems to have a hole, in particular, one of the lemmas (Lemma 6) is incorrect [42]. On the way to proving our main result on the general budgeted

allocation problem, we provide a different proof for matching. This proof has the advantage that it is simpler and direct, in that we do not consider any Refusal algorithm (like EARLY) at all, but prove the factor of Greedy via direct arguments. This simpler analysis enables us to extend the scope of the algorithm and the analysis to our general assignment problem in the randomized input model[3]. We note that the notion of Refusal (or EARLY) used in [37] does not even make sense in the general budgeted allocation problem.

Our direct proof for the case of matching follows the main ideas from [37], but also introduces some new ideas. To provide a direct proof for Greedy we use a different classification of permutations: For each boy p, we classify the set of all permutations into those in which p remains unmatched and those in which p gets matched. We further classify the latter class into two subclasses – *good* and *bad* – depending on the structure of the match. We then bound the number of unmatched boys by showing that for every unmatched boy there is a certain fraction of bad matches, and for every bad match there is a certain fraction of good matches. This gives us the factor of Greedy.

The Tagging Procedure: In going from the case of matching to the more general case of budgeted allocation, we need to develop some new techniques to take care of the complexity introduced in having different bids for the same query. The main reason is that in matching, since all the bids are 0 or 1 (with a budget of 1), any match is as good as another, while this is not the case with different bids. This creates several different (but related) issues: In matching, if a boy gets matched to a *wrong* girl then it can affect at most one other boy, in the sense that at most one other boy might not get matched. In the general problem, if a query goes to a *wrong* bidder then it can affect several other queries in many different ways. To capture

[3]Our result, including the extension to the budgeted allocation problem, precedes the communication [42].

this interaction we introduce a *'tagging procedure'* which finds an appropriate map from a query to other queries. This is required in order to define notions of good and bad events. Another difference from matching is that a query may be tagged simultaneously by any number of good and bad tags, while in matching the mapping is always either good or bad, from exactly one boy to another.

On Monotonicity: In our analysis, an extremely important property that we need is that of *monotonicity* of the allocation algorithm: If you move a query to the front of the permutation, then each bidder spends at least as much money now as before (see Lemmas 8 and 10 for details). This property seems very useful to get a handle on the analysis: Our proof in the matching case exploits the fact that Greedy is monotonic, while the proof in [37] has a hole precisely because EARLY is not monotonic. However, upon moving to the general problem, it turns out that Greedy is *not monotonic* as such. This creates an obstacle in the proof. We get around it by analyzing a *variant of Greedy*: In the case that a bidder has not yet exhausted his budget, but his bid for a query is larger than his remaining budget, we do not truncate his bid, but instead over-spend his budget. Each bidder will overspend at most once, and we will not count the overspending as part of the revenue. This variant of Greedy can be shown to be monotonic, and therefore easier to analyze. We then show that the revenue of Greedy is almost the same as that of the variant, thus proving a bound for Greedy.

Regarding our random permutation model of input, we note that it is not new, used most notably in the classic secretary-style problems [19] (see also [4]), in which a random permutation of a set of numbers arrives online and we have to pick the largest number when it arrives. Our problem is conceptually very different, since it involves *partitioning* the entire random sequence into m parts to maximize the sum of the values of the parts, rather than picking one element or structure in the sequence which is the largest. Likewise, the techniques from secretary-style problems do not

seem to apply here. The random permutations model has also been used recently in the data-streams literature [28].

We first study the simpler case of bipartite matching in Section 3.1, before analyzing the general problem in Section 3.2. The reason is that the proof in the case of matching may be easier to understand and may also help better understand the general result. We analyze the i.i.d. model in Section 3.3.2, and provide the lower bounds in the permutation model in Section 3.3.3. We conclude with a discussion in Section 5.6.

3.1 The Case of Online Bipartite Matching

In this section we consider the special case of bipartite matching (bids all 0 or 1, budgets all 1). In this problem, there is a bipartite graph (boys and girls). In the random permutations model, only the girls are known to the algorithm in advance, while the boys arrive in a random permutation. As a boy arrives, his incident edges are revealed, and he can be matched off to some girl. The Greedy algorithm matches each arriving boy to the first neighboring girl who is still available (first in some predetermined arbitrary order). We proceed to analyze the performance of Greedy in this model. By duality [37], this also gives a proof for RANKING in the online worst case model.

We label the boys and girls with numbers from 1 to n. We will use $p, q \in [n]$ to denote boys and girls, and use $s, t \in [n]$ to denote the positions in the permutation. We also use the notion of *time*: time t will denote the event when Greedy tries to allocate boy at position t. We assume that there is an optimal matching of size n, and that $\forall\ p \in [n]$ it matches boy p with the girl p. (This assumption is for simplicity of presentation, and can be removed later). Let Ω be the set of all permutations of $[n]$. For a permutation $\sigma \in \Omega$, $\sigma(s)$ will represent the boy at position s, and $\sigma^{-1}(p)$, the position of boy p.

3.1.1 Intuition:

Whether a boy p gets matched or not clearly depends on his position in the random permutation. If he is high enough in the permutation, then he will get matched, but if he comes in low in the permutation, then by the time he arrives, his desirable girls may all have been matched off to other boys. We need to analyze the relationship between the different permutations, and show that the fraction of permutations in which the boy is unmatched (averaged over the boys) is not too large.

Let us say that boy p remained unmatched in a particular permutation σ. Note that this could only have happened if girl p was already matched (say, to boy p') when boy p arrived. So we can associate the *miss* of boy p to the *match* of boy p'. Likewise, we can associate every miss to a unique match. But notice that all these *matches* that we considered are of special kind: girl p got matched to boy p' and boy p arrived *after* boy p'. We will later define these kind matches to be *bad matches*. Now let us consider the second kind of *match*: i.e. girl p got matched to boy p' and boy p arrived *before* boy p'. We shall later define such a match of boy p' to be a *good match*. Note that in a good match we are guaranteed that both boy p and girl p got matched (as well as boy p'). Our goal is to prove that, on the average over all permutations, there are many matches, in particular, many good matches. Clearly, every miss for boy p is caused by a bad match for some other boy q (who stole his girl p). This immediately shows a factor of $1/2$ for Greedy (in fact for every permutation). We will show something stronger: On average over all permutations, the misses result in a larger fraction of bad and good matches (averaged over the boys).

3.1.2 Proof Outline:

For each $\sigma \in \Omega$, $p \in [n]$ and $s \in [n]$, define:

$$miss_\sigma(s,p) = \begin{cases} 1 & \text{if } \sigma(s) = p \text{ and boy } p \text{ remains unmatched at the end of Greedy,} \\ 0 & \text{otherwise.} \end{cases}$$

$$good_\sigma(s,q) = \begin{cases} 1 & \text{if girl } q \text{ is matched to boy } \sigma(s), \text{ and } \sigma^{-1}(\text{boy } q) \leq s, \\ 0 & \text{otherwise.} \end{cases}$$

$$bad_\sigma(s,q) = \begin{cases} 1 & \text{if girl } q \text{ is matched to boy } \sigma(s), \text{ and } \sigma^{-1}(\text{boy } q) > s, \\ 0 & \text{otherwise.} \end{cases}$$

Define also the following *partitions* for every boy p. Partition the set Ω into groups of permutations s.t. in each group the relative order of all but boy p is fixed. Let Ω_p be one such group. Let $\sigma_k \in \Omega_p$ be the permutation in which boy p occurs at position k.

We will prove three main relationships between aggregates of these variables. The first inequality follows immediately from the definitions (either the boy in position t remains unmatched, or some girl gets matched to him). The other two require a detailed look at the relationship between different permutations, and their proofs are provided immediately below in Section 3.1.3. Lemmas 2 and 3 hold for every $p \in [n]$ and every group Ω_p.

Lemma 1

$$\forall\ \sigma \in \Omega, \quad \forall\ t \in [n]: \qquad \sum_p good_\sigma(t,p) + \sum_p bad_\sigma(t,p) + \sum_p miss_\sigma(t,p) \quad = \quad 1$$

Lemma 2

$$\forall\ t \in [n]: \qquad miss_{\sigma_t}(t,p) \quad \leq \quad \sum_{\sigma \in \Omega_p} \sum_{s:s<t} \frac{bad_\sigma(s,p)}{n-s}$$

Lemma 3

$$\forall t \in [n-1]: \qquad \sum_{\sigma\in\Omega_p}\sum_{s\le t} bad_\sigma(s,p)\frac{s}{n-s} \quad \le \quad \sum_{\sigma\in\Omega_p}\sum_{s:s\le t+1} good_\sigma(s,p)$$

The three lemmas above constrain the variables miss, bad and good at the end of Greedy, for every input. Note how Lemmas 2 and 3 show that if there are many misses then there are many bad matches and therefore many good matches.

Note that the revenue of Greedy is:

$$Revenue = \sum_s \left[\underset{\sigma\in\Omega}{E}[\sum_p bad_\sigma(s,p)] + \underset{\sigma\in\Omega}{E}[\sum_p good_\sigma(s,p)] \right]$$

We minimize the revenue over the constrained region given by the inequalities to get the worst performance of the algorithm (on a random permutation of the worst set Q). This program turns out to be linear, and we can solve the LP to show a factor of $1-1/e$. This technique is known as a *factor revealing LP*.

Theorem 4 *The competitive ratio of Greedy in the random permutations input model (and hence of RANKING in the online model) is* $1-1/e$.

Aggregating the inequalities. Now we will aggregate the constraints in Lemma 2 and 3 over all the permutations and girls. For that, we need to first define the following:

$$miss(s) = \underset{\sigma\in\Omega}{E}[\sum_p miss_\sigma(s,p)], \quad bad(s) = \underset{\sigma\in\Omega}{E}[\sum_p bad_\sigma(s,p)], \quad good(s) = \underset{\sigma\in\Omega}{E}[\sum_p good_\sigma(s,p)]$$

By summing the constraints in Lemma 2 over all permutations classes Ω_p and all p, we get

$$\forall t: \qquad \sum_p\sum_{\sigma\in\Omega} miss_\sigma(t,p) \le \sum_p\sum_{\sigma\in\Omega}\sum_{s:s<t} bad_\sigma(s,p)/(n-s)$$

Changing the order of summation and dividing by $|\Omega|$, we get,

Corollary 5

$$\forall t: \qquad miss(t) \leq \sum_{s:s<t} bad(s)/(n-s)$$

Similarly we can get,

Corollary 6

$$\forall t: \qquad \sum_{s \leq t} bad(s)\frac{s}{n-s} \leq \sum_{s:s \leq t+1} good(s)$$

Proof of Theorem 4 : From Lemma 1 and Corollaries 5 and 6 we have:

$$\begin{aligned}
&\text{Min: } \sum_t [bad(t) + good(t)] \\
\text{s.t.:} \qquad \forall t < n: \quad &\sum_{s \leq t+1} good(s) - \sum_{s \leq t} bad(s)\tfrac{s}{n-s} \quad \geq \quad 0 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(Corollary 6)} \\
\forall t \leq n: \quad &good(t) + bad(t) + \sum_{s<t} \tfrac{bad(s)}{n-s} \quad \geq \quad 1 \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(Corollary 5, Lemma 1)} \\
&good(t), bad(t), miss(t) \quad \geq \quad 0
\end{aligned} \tag{6}$$

Now change variables as follows: $m_t = good(t) + bad(t), \quad v_t = bad(t)/(n-t)$

$$\begin{aligned}
&\text{Min: } \sum_t m_t \\
\text{s.t.:} \quad &\sum_{s \leq t+1} m_s - n \sum_{s \leq t} v_s \quad \geq \quad 0 \\
&m_t + \sum_{s<t} v_s \quad \geq \quad 1 \\
&m_t, v_t \quad \geq \quad 0
\end{aligned}$$

It can be shown that dropping the term m_{t+1} from the first set of equations does not affect the LP solution by much. A proof of a similar statement is in KVV, and so are the calculations. We provide an LP based calculation here, for completeness. We get the following primal-dual pair:

$$\text{Min: } \sum_t m_t \qquad\qquad \text{Max: } \sum_t \alpha_t$$

s.t.: s.t.:

$$\sum_{s\le t} m_s - n\sum_{s\le t} v_s \ge 0 \qquad\qquad \sum_{s>t} \alpha_s - n\sum_{s\ge t} \beta_s \le 0$$

$$m_t + \sum_{s<t} v_s \ge 1 \qquad\qquad \alpha_t + \sum_{s\ge t} \beta_s \le 1$$

$$m_t, v_t \ge 0 \qquad\qquad \alpha_t, \beta_t \ge 0$$

Both LPs have the property that if we set all inequalities tight, we get a feasible solution. This means that these solutions are optimal.

$$m_t = \alpha_{n+1-t} = (1-1/n)^{i-1}, v_t = \beta_{n+1-t} = \tfrac{1}{n}(1-1/n)^{i-1}$$

The optimal objective function sums up to $n(1-1/e)$ proving the theorem[4].

□

3.1.3 Some Basic Properties of Greedy and Proofs of the Inequalities:

We start by describing three basic properties (Lemmas 7, 8 and 9) of the Greedy algorithm and the variables defined above. These will help up work towards our main lemmas (Lemmas 2 and 3). The first lemma follows easily from the definitions.

Lemma 7 [Prefix Property] *For any two permutations σ_1 and σ_2, and any $t \in [n]$, if we have $\forall\ s \le t:\ \sigma_1^{-1}(s) = \sigma_2^{-1}(s)$, then the runs of Greedy on σ_1 and σ_2 are identical from time 1 to time t.*

[4]We assumed that the optimal matching is perfect, i.e. $OPT = n$. The case of $OPT < n$ can be taken care of with minor changes in the proof.

The second lemma describes a crucial monotonicity property of the Greedy algorithm. Informally, it says that if you move a boy up front in the permutation, then the set of girls who used to be matched by time t are now all matched by time $t+1$. Formally, for $s \in [n]$, define $G_s(t)$ to be the set of the girls matched during the time 1 to t by Greedy in the run on permutation σ_s.

Lemma 8 [Monotonicity] $\forall\ s, m \in [n],\ s < m$:

$$(1) \qquad \forall\ t \in [1, s-1] : \quad G_m(t) = G_s(t)$$

$$(2) \qquad \forall\ t \in [s-1, m-1] : \quad G_m(t) \subseteq G_s(t+1)$$

Proof: The first part follows directly from Lemma 7. We will use induction on t to prove the second part. The base case is when $t = s-1$, and follows easily from the first part itself. Suppose the lemma holds for all $t \leq k$, where $s-1 \leq k < m-1$. We will show that the lemma holds for $t = k+1$. By hypothesis, $G_m(k) \subseteq G_s(k+1)$. Notice that the boy at position $k+1$ in σ_m is same as boy at position $k+2$ in σ_s. Let us denote this boy by b.

If b is unmatched in the run of Greedy on σ_m, then the induction step holds trivially. So, instead, assume that b is matched to girl g in the run of Greedy on σ_m. We shall show that $g \in G_s(k+2)$. Suppose not. This means that girl g was available when Greedy was matching b in σ_s, but the boy b got matched to some other girl $g' \notin G_s(k+1)$. By the induction hypothesis we have $g' \notin G_m(k)$. Therefore both g and g' were available at time $k+1$ in the run on σ_m, and Greedy should have preferred g' over g in that run as well, a contradiction. Hence $g \in G_s(k+2)$ and the lemma holds by induction. $\square$

From the definitions, we know that for any permutation σ, at most one out of the $2n$ variables, $\{good_\sigma(s,p), bad_\sigma(s,p)\}_{s\in[n]}$ can be 1:

$$\forall\ \sigma : \qquad \sum_t (good_\sigma(t,p) + bad_\sigma(t,p)) \leq 1 \tag{7}$$

The third property tells us which variable is 1, for each permutation $\sigma_x \in \Omega_p$:

Lemma 9 [Partition] *Fix p, and a partition Ω_p. Consider the run of Greedy on the permutation σ_n. If girl p gets matched to the boy at position t, for some $t \in [n]$, then:*

$$(1) \qquad \forall\ x \in [1,t]: \qquad \sum_{s:s\leq t+1} good_{\sigma_x}(s,p) = 1$$

$$(2) \qquad \forall\ x \in [t+1,n]: \qquad bad_{\sigma_x}(t,p) = 1$$

Proof: (2): By Lemma 7 (Prefix Property), we know that $\forall\ x \in [t+1,n]$, in the run on σ_x, girl p is matched to the boy at position t, and therefore $bad_{\sigma_x}(t,p) = 1$.

(1): By Lemma 8 (Monotonicity), we know that $\forall\ x \in [1,t]$, in the run on σ_x, girl p is matched to some boy whose position lies in the range $[x, t+1]$, and therefore $\sum_{s\in[x,t+1]} good_{\sigma_x}(s,p) = 1$. $\square$

Now we are ready to prove the main inequalities (Lemmas 2 and 3):

Proof of Lemma 2 : If $miss_{\sigma_t}(t,p)$ equals 0, then lemma holds trivially. So assume that $miss_{\sigma_t}(t,p) = 1$. This means that in σ_t, boy p (who is in position t) remains unmatched. This is possible only if girl p is matched to some boy whose position is some $t^* < t$. Now by Lemma 9, $\forall\ s \in [t^*+1, n]$, $bad_{\sigma_s}(t^*,p) = 1$. This gives the following sequence:

$$\begin{aligned}
\sum_{\sigma\in\Omega_p}\sum_{s:s<t} bad_\sigma(s,p)/(n-s) &\geq \sum_{\sigma\in\Omega_p} bad_\sigma(t^*,p)/(n-t^*) \qquad (t^* < t)\\
&\geq \sum_{s=t^*+1}^{n} bad_{\sigma_s}(t^*,p)/(n-t^*)\\
&= 1 \qquad\qquad\qquad\qquad (\text{Lemma 9})\\
&= miss_{\sigma_t}(t,p)
\end{aligned}$$

□

Proof of Lemma 3 : We consider two different cases: whether girl p get matched or remains unmatched in the run on σ_n.

Case 1: Girl p remains unmatched in σ_n. Take any permutation $\sigma_x \in \Omega_p$, $x \in [n]$. If girl p is matched in σ_x then, by Lemma 7 (Prefix Property), it could only be to a boy in some position greater than or equal to x. Hence, $\forall\ t: \quad bad_{\sigma_x}(t,p) = 0$, and the lemma holds trivially.

Case 2: Girl p is matched in σ_n. Suppose she is matched to the boy in position t^*. There are two subcases:

Case 2.1: $t < t^*$. Lemma 9 and Lemma 7 tell us which of the variables are 1. We see that for every $\sigma \in \Omega_p$, and every $s < t^*$, $bad_\sigma(s,p) = 0$. Hence in this case, the left hand side is 0, and the lemma follows trivially.

Case 2.2: $t \geq t^*$. Again using Lemma 9 and Lemma 7 we get the following:

$$\begin{aligned}
\sum_{\sigma\in\Omega_p}\sum_{s\leq t} bad_\sigma(s,p)\frac{s}{n-s} &= \sum_{\sigma\in\Omega_p} bad_\sigma(t^*,p)\frac{t^*}{n-t^*} \\
&= t^* \\
&= \sum_{\sigma\in\Omega_p}\sum_{s:s\leq t^*+1} good_\sigma(s,p) \\
&= \sum_{\sigma\in\Omega_p}\sum_{s:s\leq t+1} good_\sigma(s,p)
\end{aligned}$$

All the equalities follow from Lemma 9 and Equation (7). The first equality holds because $\forall\ s \neq t^*,\ \forall\ \sigma \in \Omega_p,\ bad_\sigma(s,p) = 0$. The second equality holds because $bad_\sigma(t^*,p) = 1$ for precisely the last $n - t^*$ of the $\sigma \in \Omega_p$. Similarly, the last two equalities hold because $\sum_{s:s\leq t^*+1} good_\sigma(s,p) = 1$ for precisely the first t^* of the σ in Ω_p, and the rest of the variables are 0. This proves the lemma in this case as well.

□

3.2 The General Problem

In this section we study the general assignment problem as defined in section 1. As mentioned there, our analysis holds for a larger class of inputs which includes bipartite matching and the case of small bids. We describe this class here:

For each bidder i, consider an allocation of some subset of the queries to bidder i, say $(q_1, q_2, ..., q_k)$, which straddles the budget in the following sense: $\sum_{j=1}^{k-1} bid_{ij} < B_i \leq \sum_{j=1}^{k} bid_{ij}$. For every such allocation we are guaranteed that the overspending is small compared to the budget, i.e., $\sum_{j=1}^{k} bid_{ij} - B_i \ll B_i$.

We will show that Greedy, which assigns every query that arrives to the highest bidder with a non zero remaining budget, has a competitive ratio of $(1 - 1/e)$ when the input arrives in a random permutation of the (unknown) query set (the case of i.i.d. inputs is considered in Section 3.3.2). Note that OPT is an offline optimum algorithm and will have same allocation for any such input permutation.

Our analysis of Greedy in the matching case involved a mapping between a boy p and the boy to whom girl p got matched. In the more general problem with different bids, this kind of association becomes non trivial. For every query p there is a corresponding bidder i_p to whom OPT allocated query p. But now Greedy could assign several queries with different bid values to this bidder and the mapping between query p and these queries is not straightforward. Our tagging procedure takes care of this difficulty:

For a permutation $\sigma \in \Omega$, $\sigma(s)$ will represent the query at position s, and $\sigma^{-1}(p)$, the position of query p.

The Tagging Procedure: Fix an optimal allocation OPT and also fix, for each bidder, an arbitrary *ordering* of the queries assigned to it in OPT. Now consider any query q and the bidder i to whom OPT assigned q. In the fixed ordering, when OPT assigned q to i, suppose that the money spent by i increased from x to y. Then we

will call $[x, y]_i$, the *Opt-Interval* of the query q: i.e., define $\mathit{Opt\text{-}Interval}(q) = [x, y]_i$. Greedy, run on some permutation σ, makes its own allocation, leading to a similar $\mathit{Alg\text{-}Interval}_\sigma(q)$ for each query q.

For a query q, the money in $\mathit{Opt\text{-}Interval}(q)$ is spent by Greedy on some other queries, say, e.g., q_1, q_2, q_3. We say that q_1, q_2 and q_3 *tag* the *Opt-Interval* of q. We define $tagset_\sigma(q_1, q)$ as the *set* $\mathit{Opt\text{-}Interval}(q) \cap \mathit{Alg\text{-}Interval}_\sigma(q_1)$. The same set is also called $goodset_\sigma(q_1, q)$ or $badset_\sigma(q_1, q)$, depending on whether $\sigma^{-1}(q) \leq \sigma^{-1}(q_1)$ or $\sigma^{-1}(q) > \sigma^{-1}(q_1)$.

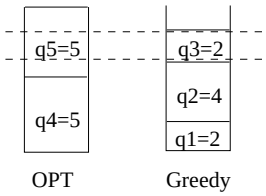

Figure 1: How the money is spent in OPT and in the run of Greedy.

Example: For illustration purposes, consider the following scenario. Focus on five special queries q_1, q_2, q_3, q_4, q_5, and a bidder i who has a budget of 10. Fix an OPT which assigns q_4 and q_5 to bidder i in that order (see figure 1). Consider a permutation σ in which the relative positions of these queries is $(q_1, q_2, q_4, q_5, q_3)$, and suppose that Greedy assigns q_1, q_2, q_3 to bidder i (in that order). The bids of the different queries are shown in the figure. Then we have the following tagsets: $tagset_\sigma(q_1, q_4) = [0, 2]_i$, $tagset_\sigma(q_2, q_4) = [2, 5]_i$, $tagset_\sigma(q_1, q_5) = \emptyset$, $tagset_\sigma(q_2, q_5) = [5, 6]_i$, $tagset_\sigma(q_3, q_4) = \emptyset$, $tagset_\sigma(q_3, q_5) = [6, 8]_i$. The first four are *badsets* and the last two are *goodsets* (since q_3 is after q_4 and q_5 in the permutation). Note also that the subinterval $[8, 10]_i$ of $\mathit{Opt\text{-}Interval}(q_5)$ is not tagged. This contributes towards the loss of Greedy, and we will define it later as a miss of q_5.

Note how a query's *Opt-Interval* can be tagged simultaneously by any number of other queries, and simultaneously with good and bad tags (unlike in the case of

matching, in which the tags were one-to-one). Also, if a bidder spends more money in Greedy compared to in OPT then some part of $\textit{Alg-Interval}_\sigma(q)$ may not even be used to tag any *Opt-Interval*. Define $\textit{extra}_\sigma(q) = \textit{Alg-Interval}_\sigma(q) \setminus \cup_p \textit{Opt-Interval}(p)$.

Define $|[x,y]_i| = y - x$. Also, for any query q, we define $\textit{opt}(q) = |\textit{Opt}-\textit{Interval}(q)|$ and $alg_\sigma(q) = |\textit{Alg} - \textit{Interval}_\sigma(q)|$. By convention, $|\emptyset| = 0$.

3.2.1 The Main Properties of Greedy and the Analysis

Recall from Section 3.1 that in the case of matching, Greedy has three properties (namely **Prefix**, **Monotonicity** and **Partition**) which we exploited in proving the relationships between the variables *miss, bad* and *good*. In the general case, as we saw above, Greedy behaves in a much less controlled manner because of the different bids. However, we manage to prove that the corresponding properties still hold, in spite of this different behavior[5]. This is sufficient to prove our main lemmas (Lemmas 13, 14 and 17), and prove a factor of $1 - 1/e$.

We

Let us first explicitly spell out a very basic property of Greedy (which we took for granted in matching):

Consistent Tie-Breaking: Consider a query q, and any two permutations σ, σ'. Define $\textit{highest}_\sigma(q)$ as the set of bidders who have the highest bid for q among all bidders with non-zero remaining budget, when q arrives during the run on σ. Define $\textit{highest}_{\sigma'}(q)$ similarly. Suppose that $\textit{highest}_{\sigma'}(q) \subseteq \textit{highest}_\sigma(q)$, and suppose that in the run on σ, q is allocated to bidder $i^* \in \textit{highest}_{\sigma'}(q)$. Then it has to be true that q is allocated to i^*, even in the run on σ'.

We now proceed to prove the monotonicity property.

Fix a query $p \in Q$. Partition the set of all permutations Ω into subsets such that

[5]As stated in the Introduction, Greedy as such is not monotonic, but we modify Greedy so that it does not truncate the bid by the remaining budget. This variant is the one we analyze here, and is monotonic.

for each subset, all the permutations within the subset have the same order on all queries, except for query p. For any subset Ω_p of this partition, let the permutations be called $\sigma_1, .., \sigma_n$, according to the position of p.

Define $alg_\sigma(i,t)$ to be the amount of money that bidder i has spent in the run of Greedy on the permutation σ[6] up to (just after) time t. We have the following version of the **monotonicity lemma**:

Lemma 10 [Monotonicity] *For every $s, s' \in [1,n]$, $s' < s$, for every $t \leq s-1$, and for every bidder $i \in [1,n]$, we have one of the two conditions:*

Either: $alg_{\sigma_{s'}}(i, t+1) \geq alg_{\sigma_s}(i,t)$

Or: $alg_{\sigma_{s'}}(i, t+1) \geq B_i$

Proof: Fix $s' < s \leq n$ and a bidder i. We will prove the lemma by induction on t going from 1 to $s-1$. We are comparing the run of ALG on $\sigma_{s'}$ and on σ_s up to time t. Note, firstly, that the two permutations are identical until position $s'-1$, hence so are the two runs (by the Prefix Property). So, in fact, $\forall\, t < s'$, $alg_{\sigma_{s'}}(i,t) = alg_{\sigma_s}(i,t)$ and so $alg_{\sigma_{s'}}(i, t+1) \geq alg_{\sigma_s}(i,t)$. Hence we may assume that the base case of the induction is $t = s'-1$. Suppose the statement is true for some $t \in [s'-1, s-2]$. We will show it is true for $t+1$ as well. This would prove the lemma.

Let q be the query $\sigma_s(t+1)$. Since $t \in [s'-1, s-2]$, we know that $\sigma_{s'}(t+2) = q$ as well. Suppose that in the run on σ_s, ALG allocates q to bidder j. We ask: where does ALG allocate q in the run on $\sigma_{s'}$? Define, for a permutation π and a time τ, $avail_\pi(\tau)$ as the set of bidders available with non-zero remaining budget in the run of ALG on π at (just before) time τ. By the induction hypothesis, $avail_{\sigma_{s'}}(t+2) \subseteq avail_{\sigma_s}(t+1)$. Also, we know that $j \in avail_{\sigma_s}(t+1)$ and that j is the highest bidder among bidders in $avail_{\sigma_s}(t+1)$.

[6]Note that since the algorithm may overspend budgets, $alg_\sigma(i,t)$ could be greater than B_i (however it is at most $B_i + \epsilon_i$).

Now consider two cases: If $j \in avail_{\sigma_{s'}}(t+2)$ then, by the argument above, j is the highest bidder in $avail_{\sigma_{s'}}(t+2)$ as well, and (by the Consistent Tie-breaking Property), ALG will allocate q to j in $\sigma_{s'}$. In the second case, when $j \notin avail_{\sigma_{s'}}(t+2)$, then $alg_{\sigma_{s'}}(j, t+1) \geq B_j$ anyway. Hence, both actions keep the induction statement true for $t+1$.

□

The prefix property holds as before, by the definition of Greedy:

Lemma 11 [Prefix Property] *For any two permutations σ_1 and σ_2, and any $t \in [n]$, if we have $\forall\ s \leq t :\ \sigma_1^{-1}(s) = \sigma_2^{-1}(s)$, then the runs of Greedy on σ_1 and σ_2 are identical from time 1 to time t.*

For any position s, we will use $tagset_\sigma(s,p)$ to mean $tagset_\sigma(\sigma(s),p)$. Similarly we define $goodset_\sigma(s,p)$ and $badset_\sigma(s,p)$. Also define $extra_\sigma(s) = extra_\sigma(\sigma(s))$. We get the following version of the partition lemma:

Lemma 12 [Partition] *Fix p, and a partition Ω_p.*

$$(1) \quad \forall\ x \in [1,t] : \qquad \cup_{s:x\leq s\leq t+1}\, goodset_{\sigma_x}(s,p) \supseteq tagset_{\sigma_n(t,p)}$$

$$(2) \quad \forall\ x \in [t+1,n] : \qquad badset_{\sigma_x}(t,p) = tagset_{\sigma_n}(t,p)$$

Proof: (2): Follows directly from Lemma 11.

(1): Suppose i^* be the bidder to which OPT allocated query p. By the Prefix Property, $alg_{\sigma_k}(i^*, k-1) = alg_{\sigma_n}(i^*, k-1)$, and by the monotonicity lemma either $alg_{\sigma_k}(i^*, s+1) \geq B_{i^*}$ or $alg_{\sigma_k}(i^*, s+1) \geq alg_{\sigma_n}(i^*, s)$. So either $alg_{\sigma_k}(i^*, s+1) - alg_{\sigma_k}(i^*, k-1) \geq alg_{\sigma_n}(i^*, s) - alg_{\sigma_n}(i^*, k-1)$, or $alg_{\sigma_k}(i^*, s+1) \geq B_{i^*}$. In either case, $tagset_{\sigma_n}(s,p) \subseteq \bigcup_{k\leq s'\leq s+1} tagset_{\sigma_k}(s',p)$. But recall that $tagset_{\sigma_k}(s',p)$ contributes towards $goodset_{\sigma_k}(s',p)$ for all s' s.t. $s' \geq k$. This proves the lemma. □

Define $good_\sigma(s,p) = |goodset_\sigma(s,p)|$ and $bad_\sigma(s,p) = |badset_\sigma(s,p)|$. Also define:

$$miss_\sigma(s,p) = \begin{cases} max(0, opt(p) - alg_\sigma(p)) & \text{if } \sigma(s) = p \\ 0 & \text{otherwise.} \end{cases}$$

These three basic properties shown above (Lemmas 10, 11 and 12) suffice to prove the required relationships between $good, bad$ and $miss$, and to prove the main theorem. We state these inequalities below (Lemmas 14 and 17 hold for every $p \in Q$ and every group Ω_p).

Lemma 13

$\forall\ s \in [n],\ \forall\ \sigma \in \Omega :$

$$\sum_p bad_\sigma(s,p) + \sum_p good_\sigma(s,p) + |extra_\sigma(s)| + \sum_p miss_\sigma(s,p) \quad \geq \quad opt(\sigma(s))$$

Proof: For every σ, and every $s \in [1,n]$,

$$\begin{aligned} \sum_p [bad_\sigma(s,p) + good_\sigma(s,p)] &= \sum_p |tagset_\sigma(\sigma(s),p)| \\ &= alg_\sigma(\sigma(s)) - |extra_\sigma(s)| \end{aligned} \tag{8}$$

The first equality is because of the fact that p is either above or below s in σ. The second equality follows from the definition of $extra_\sigma(s)$. Now, by definition, $miss_\sigma(s,p)$ is possibly non-zero only for $p = \sigma(s)$, in which case it is equal to $\max\{0, opt(p) - alg_\sigma(p)\}$. Hence,

$$\sum_p miss_\sigma(s,p) = \max\{0, opt(\sigma(s)) - alg_\sigma(\sigma(s))\}$$

From (8), we get that

$$\sum_p miss_\sigma(s,p) = \max\left\{0, opt(\sigma(s)) - \sum_p bad_\sigma(s,p) - \sum_p good_\sigma(s,p) - |extra_\sigma(s)|\right\}$$

thus proving the lemma. □

Lemma 14

$$\forall\ t \in [n]: \qquad miss_{\sigma_t}(t,p) \quad \leq \quad \sum_{\sigma \in \Omega_p} \sum_{s<t} \frac{bad(s,p)}{n-s}$$

Proof: Define the quantity $x := \sum_{s<t} bad_{\sigma_n}(s,p)$. By the definition of $bad_{\sigma_n}(s,p)$, $x \leq opt(p)$. In fact, we shall show the following inequality:

$$miss_{\sigma_t}(t,p) \ \leq \ x \ = \ \sum_{\sigma \in \Omega_p} \sum_{s<t} \frac{bad_{\sigma}(s,p)}{n-s} \tag{9}$$

We prove Equation (9) in a sequence of two claims:

Claim 15 $miss_{\sigma_t}(t,p) \ \leq \ x$

Proof: Let i_p^* be the bidder to whom OPT allocated the query p. We show the following subclaim:

Subclaim: Consider the run on the permutation σ_t. At time t (when ALG considers assigning query p), the remaining budget of i_p^* is at least $opt(p) - x \geq 0$.

Proof: First note that, by the Prefix Property, $\sum_{s<t} bad_{\sigma_t}(s,p) = \sum_{s<t} bad_{\sigma_n}(s,p)$ $= x$. Since in σ_t, $\forall s < t$, $bad_{\sigma_t}(s,p) = |tagset_{\sigma_t}(\sigma_t(s),p)|$, therefore $\sum_{s<t} |tagset_{\sigma_t}(\sigma_t(s),p)|$ $= x$. But this means that at least $opt(p) - x$ length of $Opt\text{-}Interval(p)$ is untagged after time $t-1$ in the run on σ_t. Of course, this means that at least $opt(p)-x$ amount of budget of i_p^* is still unspent at this time. This proves the subclaim.

If $x < opt(p)$, then using the fact that the algorithm is greedy (and that we do not truncate the bids of bidders by their remaining non-zero budgets), we see that $alg(p) \geq opt(p)$. If $x = opt(p)$, then $alg(p) \geq opt(p) - x$, trivially. This proves the claim.

Claim 16 $\sum_{\sigma \in \Omega_p} \sum_{s<t} \frac{bad_{\sigma}(s,p)}{n-s} \ = \ x$

Proof: Fix an $s < t$, and consider the value of $\sum_{\sigma \in \Omega_p} \frac{bad_\sigma(s,p)}{n-s}$. Note that, by the definition of $bad_\sigma(s,p)$ and by the Prefix Property:

$$bad_{\sigma_k}(s,p) = \begin{cases} bad_{\sigma_n}(s,p) & \text{for } k \in [s+1,n] \\ 0 & \text{for } k \in [1,s] \end{cases}$$

Hence $\frac{\sum_{\sigma \in \Omega_p} bad_\sigma(s,p)}{n-s} = bad_{\sigma_n}(s,p)$. Hence, $\sum_{\sigma \in \Omega_p} \sum_{s<t} \frac{bad_\sigma(s,p)}{n-s} = \sum_{s<t} bad_{\sigma_n}(s,p)$ $= x$.

This concludes the proof of Equation (9), and hence of the lemma.

□

Lemma 17

$$\forall\ t \in [n-1]: \quad \sum_{\sigma \in \Omega_p} \sum_{s \le t} bad(s,p) \frac{s}{n-s} \quad \le \quad \sum_{\sigma \in \Omega_p} \sum_{s \le t+1} good(s,p)$$

Proof: To do a tight counting for the proof of the lemma, we need to define a notion of weighted set. Given a universe U of elements, a weighted set S_w has weight on all its elements. In particular, an unweighted set S can be viewed as a weighted set with weight one on elements which are present in the set, and zero otherwise. We shall also define two new operations $*$ and $\oplus$ on these weighted sets. Given a weighted set S, and a number x, we define new weighted set $S' = S * x$ to mean that $weight_{S'}(e) = weight_S(e) * x$, $\forall e \in U$. Also given weighted sets S_1 and S_2, we define $S' = S_1 \oplus S_2$ to mean that $weight_{S'}(e) = weight_{S_1}(e) + weight_{S_2}(e)$. $S_1 \subseteq_w S_2$ means that $weight_{S_1}(e) \le weight_{S_2}(e)$, $\forall e \in U$. Define the measure of weighted set S, $\mu(S) = \int_U weight_S(u)du$. For the rest of the proof, the term set will mean a weighted set. We will consider sets on the universe $Opt - Interval(p)$, which, recall, is a interval of size $opt(p)$, corresponding to a query p.

Fix an $s \le t$. Consider a subset Ω_p s.t. $badset_{\sigma_n}(s,p) \neq \emptyset$. By the Prefix Property, $badset_{\sigma_i}(s,p) = badset_{\sigma_n}(s,p)$, $\forall\ i \in [s+1,n]$. By Partition lemma, Lemma 12, we

see that for every $k \leq s$, $\forall l \in [s+1, n]$: $badset_{\sigma_l}(s,p) \subseteq \bigcup_{s'=k}^{s+1} goodset_{\sigma_k}(s',p)$, where the union is a disjoint union.

Since the union is disjoint, this is the same as

$$\forall k \in [1,s],\ \forall l \in [s+1,n]: \qquad badset_{\sigma_l}(s,p)\ \subseteq_w \bigoplus_{s'=k}^{s+1} goodset_{\sigma_k}(s',p)$$

By averaging, we see that:

$$\forall\ s: \quad \frac{1}{n-s} \bigoplus_{l=s+1}^{n} badset_{\sigma_l}(s,p)\ \subseteq_w\ \frac{1}{s} \bigoplus_{k=1}^{s} \bigoplus_{s'=k}^{s+1} goodset_{\sigma_k}(s',p)$$

or,

$$\forall\ s: \quad \bigoplus_{l=s+1}^{n} badset_{\sigma_l}(s,p)\frac{s}{n-s}\ \subseteq_w\ \bigoplus_{k=1}^{s} \bigoplus_{s'=k}^{s+1} goodset_{\sigma_k}(s',p)$$

But since $badset_{\sigma_i}(s,p) = \phi,\ \forall\ i \in [1,s]$ and $\forall\ s' < k:\ goodset_{\sigma_k}(s',p) = \phi$, we get:

$$\forall\ s: \quad \bigoplus_{l=1}^{n} badset_{\sigma_l}(s,p)\frac{s}{n-s}\ \subseteq_w\ \bigoplus_{k=1}^{s} \bigoplus_{s'=1}^{s+1} goodset_{\sigma_k}(s',p)$$

And again, since $\forall\ k > s, s' \leq s:\quad goodset_{\sigma_i}(s',p) = \phi$, we get :

$$\forall\ s: \quad \bigoplus_{l=1}^{n} badset_{\sigma_l}(s,p)\frac{s}{n-s}\ \subseteq_w\ \bigoplus_{k=1}^{n} \bigoplus_{s'=1}^{s+1} goodset_{\sigma_k}(s',p)$$

or,

$$\forall\ s: \quad \bigoplus_{\sigma\in\Omega_p} badset_{\sigma}(s,p)\frac{s}{n-s}\ \subseteq_w\ \bigoplus_{\sigma\in\Omega_p} \bigoplus_{s'=1}^{s+1} goodset_{\sigma}(s',p)$$

Also, by definition, $\forall\ u \neq s,\ u,s \leq t$: $badset_{\sigma_n}(s,p) \cap badset_{\sigma_n}(u,p) = \emptyset$. Hence:

$$\bigoplus_{s\leq t} \bigoplus_{\sigma\in\Omega_p} badset_{\sigma}(s,p)\frac{s}{n-s}\ \subseteq_w\ \bigoplus_{\sigma\in\Omega_p} \bigoplus_{s\leq t+1} goodset_{\sigma}(s,p)$$

Therefore,

$$\mu\left(\bigoplus_{s\leq t} \bigoplus_{\sigma\in\Omega_p} badset_{\sigma}(s,p)\frac{s}{n-s}\right) \leq\ \mu\left(\bigoplus_{s\leq t+1} \bigoplus_{\sigma\in\Omega_p} goodset_{\sigma}(s,p)\right)$$

$$\sum_{s\leq t}\sum_{\sigma\in\Omega_p} \mu(badset_{\sigma}(s,p))\frac{s}{n-s}\ \leq\ \sum_{s\leq t+1}\sum_{\sigma\in\Omega_p} \mu(goodset_{\sigma}(s,p))$$

$$\sum_{s\leq t}\sum_{\sigma\in\Omega_p} bad_{\sigma}(s,p)\frac{s}{n-s}\ \leq\ \sum_{s\leq t+1}\sum_{\sigma\in\Omega_p} good_{\sigma}(s,p)$$

□

Theorem 18 *The factor of Greedy in the random-permutation input model is at least* $(1 - 1/e)$.

Sketch of Proof: We shall use the same definition of $miss(s)$, $bad(s)$ and $good(s)$ as defined in the proof of theorem 4. Let $extra(s) = E_{\sigma \in \Omega}|extra_\sigma(s)|$. Using similar approach as in the proof of theorem 4, we will get the following three corollaries from Lemmas 13, 14 and 17.

Corollary 19

$$\forall t: \qquad bad(t) + good(t) + extra(t) + miss(t) \geq OPT/n$$

Corollary 20

$$\forall t: \qquad miss(t) \leq \sum_{s:s<t} bad(s)/(n-s)$$

Corollary 21

$$\forall t: \qquad \sum_{s \leq t} bad(s)\frac{s}{n-s} \leq \sum_{s:s \leq t+1} good(s)$$

Using the above corollaries, we get the following LP:

$$\text{Min: } \sum_t [bad(t) + good(t) + extra(t)]$$

$$\text{s.t.:} \qquad \forall t < n: \quad \sum_{s \leq t+1} good(s) - \sum_{s \leq t} bad(s)\tfrac{s}{n-s} \geq 0$$

$$\forall t \leq n: \quad good(t) + bad(t) + extra(t) + \sum_{s<t} \tfrac{bad(s)}{n-s} \geq OPT/n$$

$$good(t), bad(t), extra(t) \geq 0$$

First inequality can in fact be relaxed to $\forall t < n: \quad \sum_{s \leq t+1}(good(s) + extra(s)) - \sum_{s \leq t} bad(s)\frac{s}{n-s} \geq 0$.

Now substituting m_t for $good(t)+bad(t)+extra(t)$, v_t for $bad(t)/n-t$, and solving it similar to online bipartite matching, we get that the value of objective function is at least $OPT(1 - 1/e)$.

Recall, however, that we had modified Greedy to allow overspending of budgets. We can show (e.g., by induction) that the revenue of this variant of Greedy is almost the same as that of the pure Greedy, which truncates the bids by the remaining budget, when the bids are small (each bidder's spending may be different by at most $n\epsilon$, where epsilon is the largest bid). Thus we have proved the factor for both versions of Greedy.

3.3 Additional Results

3.3.1 Tightness of the analysis

We prove that our analysis is tight, by giving an example in which Greedy does no better that $1 - 1/e$:

Theorem 22 *The factor of Greedy in the random-permutation input model is no more than* $(1 - 1/e)$.

In this instance, there are N bidders, each with a budget of L (a large integer). The queries are grouped into N groups, $Q_1, Q_2, ..., Q_N$. Each group has L identical queries. Each query in Q_i gets the following bids: Bidders 1 through $i - 1$ bid 0; bidders i through N bid 1. Clearly, $OPT = LN$, by allocating, for all i, all the queries in Q_i to bidder i.

We will assume that Greedy breaks ties in the following way: when an query q arrives, it will be allocated to the highest numbered bidder who bids 1 and has available budget. The assumption about bad tie-breaking can be removed by a simple perturbation of the bids: for example, for all i, replace all the 1-bids of bidder i by $1 + \epsilon_i$, where $0 < \epsilon_1 < \epsilon_2 < \cdots < \epsilon_N \ll 1$. The budgets can be adjusted accordingly.

Now consider the performance of Greedy on the sequence of these queries arriving in a random permutation σ. Clearly, the first L queries in σ will all be allocated to bidder N (who bids for all queries). Among the next L queries in σ, all will be

allocated to bidder $N-1$, except for any queries from Q_N, which are left unallocated. Now among the third L queries in σ, some are allocated to $N-1$, most to $N-2$ and some are lost. Let us instead count in a different manner: It is clear that bidder N, by symmetry, is allocated at most L/N queries from Q_j, in expectation, $\forall\ j$. Similarly, bidder $N-1$, by symmetry, is allocated at most $L/(N-1)$ queries from Q_j, in expectation, $\forall\ j \leq N-1$. In this manner one can see that bidder i is allocated at most L/i queries from Q_j, in expectation, $\forall\ j \leq\ i$.

Counting differently, for $i = 1, .., N$, the number of queries in Q_i that get allocated is $\min\{L, \sum_{j \leq i} \frac{L}{N-j}\}$. Summing this up, we can show that the expected amount of money spent by Greedy is $LN(1-1/e) = (1-1/e)OPT$.

We note that this analysis is very similar to that in [48] for the lower bound in the worst case online setting, the two being a form of duals of each other. The difference is that the role of bidders and query groups is reversed – here it is the last $N(1-1/e)$ bidders who finish most of their budgets, while in [48] it was the first $N(1-1/e)$ query groups that get almost completely allocated.

3.3.2 The *i.i.d.* randomized input model

In this model there is a fixed (but unknown) distribution on Q, and the next query in the sequence is picked independently from this distribution. A simple reduction to our random permutation setting shows that:

Theorem 23 *The factor of Greedy in the independent distribution input model is at least* $(1-1/e)$. *Also, there is an example distribution over which Greedy does no better than* $1-1/e$.

Proof:(Sketch) In this model we have a sample space consisting of the different query sequences obtained by drawing n times from the given distribution. We have to compare the expected performance of Greedy with the expected performance of

OPT over this sample space[7]. Divide the sample space into classes s.t. the *set* of queries is the same for every sequence in a class. Clearly, each class consists of all the permutations of some set, and furthermore, the probability of occurrence of each sequence in a class is the same. Suppose Ω' is one such class. Then, by Theorem 18 we get that $E[GREEDY|\Omega'] \geq (1-1/e)E[OPT|\Omega']$. Taking expectation over the different classes, we get the first part of the theorem.

For the second part of the theorem, we give a tight example – suppose there are N bidders, and we are given a uniform distribution over N keywords, where the i^{th} keyword gets the following bids: Bidders 1 through $i-1$ bid 0; bidders i through N bid 1. We will use the tie breaking rules as discussed in the proof of theorem 22. We will sample LN queries from the above distribution. Now by the Chernoff bound, we can show that, for all δ, there is a large enough L so that w.h.p. each keyword is sampled at most $L+\delta/N$ times. Now take the sample space in which each keyword occurs at least $L-\delta/N$ times, and divide it into families s.t. in each family, every sequence has a fixed set of queries. Using arguments similar to the ones in the proof of theorem 22, we can show that expected performance of greedy, conditional over any such family, is at most $LN(1-1/e)+\delta$. Now taking expectation over all such families we get that expected performance of greedy is at most $LN(1-1/e)+\delta$.

□

3.3.3 Lower Bound

The proof of the following theorem uses a uniform distribution on a set of simple 3×3 size bipartite matching examples.

Theorem 24 *No deterministic algorithm can have a competitive ratio better than* $3/4$*, and no randomized algorithm can have a competitive ratio better than* $5/6$*, for*

[7]The result also holds for the average of the ratios of Greedy and OPT, and not only for the ratio of their averages.

the case of bipartite matching in the random permutations model.

(Sketch of Proof) For the bound on deterministic algorithms, consider a very simple 2 boy, 2 girl setting, where boy 1 could be matched to both the girls, but boy 2 can only be matched to one of the two girls. By looking at the "code" of the deterministic algorithm, we can decide which girl boy 2 can be matched to, so that in one of the two random permutations of the boys, only one pair can be matched. This shows a factor of 3/4.

For the lower bound on randomized algorithms, we will use Yao's Lemma [66] to prove this statement. We start with a distribution on inputs of the following form: Take an $n \times n$ complete upper triangular matrix with $0, 1$ entries. This is the incident matrix of the bipartite graph, with the rows being the girls and columns being boys. We use the uniform distribution over different inputs corresponding to all permutations of the row names. As according to the model, each input has the columns arrive in a random permutation. Thus the distribution is essentially uniform distribution on both row and column permutations of the complete upper triangular matrix.

As opposed to the proof in [37] for the worst case input model, it turns out to be difficult to characterize the performance of *any* deterministic algorithm on this input distribution. This is because a deterministic algorithm can (in this model) be smart and use a lot of information from the past (as a very simple example, if it sees a column with $n - 1$ 1's and then a column with n 1's then it knows that the extra row is the row with n 1's). We have not been able to come up with a general analysis for n, but we can do a brute force analysis for $n = 3$, in which we can take care of how any deterministic algorithm may use all such information from the past. The brute force analysis shows that the performance of any deterministic algorithm over this input distribution is no more than 5/6, thus proving a lower bound of 5/6 for randomized algorithms in the random permutations model.

3.4 Further Discussion

Our motivation for studying Greedy was that this is what is actually used. However, the question naturally arises: Is Greedy optimal in our distributional model? If we make the assumption that there are n types of keywords, and many queries of each type, and that we know the length of the query sequence, then there is an allocation algorithm with a ratio very close to 1: Sample the sequence for an ϵ fraction of its length and learn the distribution of queries. Now solve and round a revenue maximization Linear Program to get $1 - \epsilon$ ratio allocation. However, this sampling method will not work if all the queries are distinct, and of course, this method could be arbitrarily bad in the worst case. We do not know of any algorithm provably better than Greedy in the random permutation model, but we believe that the algorithm from [48] performs strictly better. Currently we can show that it cannot do better than a factor $\alpha \sim 0.81$.

Recall that in the case of matching, it was proved in [37] that RANKING in the worst case and Greedy in the random permutations case are duals of each other. The question naturally arises for the general assignment problem: What is the dual algorithm to Greedy? It turns out that no such duality result seems to hold in the general case. Furthermore any such dual would have the bidders arriving in online, and hence would not apply to online assignment. We note that a attempt was made in [48] towards this, but that proof was incorrect and was withdrawn.

Finally, it is important to investigate the extent to which the RANKING algorithm of [37] and its ideas can be generalized, maybe even to offline algorithms.

CHAPTER IV

BUDGETED AUCTIONS: DECREASING VALUATION BIDS

In this chapter, we study a new bidding model for the sponsored search auctions (budgeted auction with small bid/budget ratio) – *decreasing valuation bids*. This provides a richer language than the current model for advertisers to convey their preferences. Furthermore, we show that the competitive ratio of online algorithms that have been studied in the standard setting is retained.

Bidding language is one of the most important design parameters in auction design. This is the *interface* provided to the bidders by the seller, which allows them to express their preferences to the seller. There is always a trade-off in this choice. The more complex a bidding language is, the better can it capture bidder preferences, and indirectly, the better it is for the seller. But at the same time, it is essential to have a simple bidding language, so that the bidders will be able to translate their innate preferences into the language in the first place. Simplicity of expression is not the only reason preventing us from choosing highly complex bidding languages. A second reason is *computational*: Even if the bidders express their preferences in a complex bidding language, it may be impossible for the seller to process such complex preferences and decide on an optimal (or even a good) outcome efficiently. Such a trade-off becomes apparent in the design of complex auctions, and has been studied in detail in the literature, for example, in the case of combinatorial auctions [51].

The bidding language currently provided by the search engines is essentially of the following form: A bidder i can bid, for each keyword q he is interested in, a monetary bid b_{iq}, expressing the value he gets if his ad is displayed with search results for queries

of type q. Together with the bids, the bidder is also allowed to report a *daily global budget* B_i, which means that over all the keywords, he is not willing to spend more than B_i.

While this bidding model does capture the essential features of advertiser preferences, namely, the individual bids and a global budget, it fails to express more complex constraints that the advertisers may have. For example, the advertiser may not want to be in a situation in which he wins many ad slots, but all of them for queries of the same single keyword. He would prefer to have diversity, in the sense of winning a reasonable amount of ad slots for several different keywords of his choice. There are two reasons to expect real advertisers to have such preferences: firstly, advertisers may wish to make their presence felt in several different sub-markets at the same time, and sell different products at comparable rates. Secondly, there are situations in which advertisers have decreasing marginal utility from subsequent advertisements, e.g., once the ad reaches a certain fraction of the target audience.

It is important to find expressive, yet simple and practically implementable bidding languages which would provide bidders with more control to express such preferences. Note that it is not even possible to simulate such preferences in the current bidding model, say by splitting into several different accounts. We stress the following four **properties of our model:** *Firstly*, our bidding model is an *add-on* to the current model, and hence can be gradually introduced on top of the current model. Bidders may choose to continue bidding as they did in the current model if they prefer. *Secondly*, as our results show, there may be no need to change the allocation algorithms used by the search engine, even upon introducing the new bidding model. *Thirdly*, the better expressivity will, in our opinion, allow the bidders to bid with more control and less risk, and therefore more aggressively, indirectly improve the revenue of the search engine. *Finally*, we believe that this model may lead to less fluctuations in the bids, as opposed to the current model in which bidders may

dynamically change their bids as they win certain number of queries.

4.1 A new bidding language: Decreasing valuation bids

In the *decreasing valuations bid* model, bidder i bids the following:

- A global daily budget B_i.
- For each keyword w he is interested in, a *decreasing function* $f_{i,w} : Z^+ \rightarrow \mathbb{R}$, which is to be interpreted as follows: If bidder i has already been allocated x number of queries of keyword w, then his bid for the next query of keyword w is $f_{iw}(x)$.

Note that the current model only allows constant functions $f_{iw}(x) = b_{iw}, \forall x$. A simple and practical special case of our model is one which allows only functions of the form

$$f_{iw}(x) = \begin{cases} b_{iw} & \text{if } x \leq t_{iw} \\ 0 & \text{otherwise} \end{cases}$$

This special case means that bidder i values each of the first t_{iw} queries of keyword w at b_{iw} each, but does not want more than t_{iw} of w's . We shall call this special case the case of q-budgets.

Remark: The notion of *spending constraints* was introduced in [17] in the context of bidder utilities in markets and the computation of market equilibrium prices. There the utility of a bidder for the next item of a type of good depended on how much *money* he had already spent on that type. Here our definition is in terms of how many *items* of that type have been allocated to the bidder. But note that in our case the price of each item is static, determined by the bid curve, so our bid curve can be simply translated to a curve in the axis of money spent instead.

4.2 Results in the new models

Better expressivity is clearly better for the bidder (as long as the language remains simple enough to understand). So with the introduction of this new models of bidding

languages, the question which arises naturally is: How does this affect the search engine's profits?

Intuitively, it is clear that the bidders will now be able to bid with more control and therefore face less risk, and will bid more aggressively. This is clearly better for the search engine, in terms of the optimal profit (OPT) derivable from the bidders. But what if the bidding language introduces computationally difficult problems for the search engine? Then it will not be able to efficiently extract a good portion of the OPT as profit. We show that our models do not introduce such computational difficulties, by describing optimal allocation algorithms in both these settings, whose competitive ratio (in an online competitive analysis model) is $1 - 1/e$, as good as that of the optimal algorithm [48] in the standard model.

We also show that our bidding language is at the correct trade-off point between expressivity, simplicity and computational efficiency. Simple generalizations of our bidding model (by adding more expressivity) result in computational problems for which no algorithms can perform better than a factor of $1/2$, and for which the natural Greedy algorithm has an arbitrarily bad factor.

4.2.1 Our Techniques

The algorithm we analyze in the new model is precisely the algorithm from [48] for the standard bidding model. For each arriving query, this algorithm (which we will call MSVV in the sequel) determines the *effective bid* of each bidder as his bid for the query scaled by a function ψ of the fraction of budget that the bidder has spent so far (the function is $\psi(x) = 1 - e^{-(1-x)}$). Then the algorithm awards the query to the bidder with maximum effective bid. It is shown in [48] that this algorithm has a competitive ratio of $1 - 1/e$ in the standard bidding model, and that this is optimal (even over randomized algorithms). We show that the same algorithm has the same factor even in our generalized bidding models. Clearly it is optimal since our models

are more general than the standard model.

Our proof technique follows the proofs in [48]. In that proof, the main idea was to show that for each query q, the algorithm gets some effective amount of money (which is the real money scaled by some factor depending on ψ) which is comparable to an effective amount that OPT gains for q. This kind of query-by-query analysis is not possible here, since the bid of a bidder i for a query q itself depends on how many other queries of that type have already been allocated to him. Thus the bid depends on the *context* in which q arrives with respect to the previous choices of the algorithm. We take care of this by a careful charging argument: we demonstrate the existence of a map between the queries that OPT assigns and the queries that ALG assigns (not necessarily to the same bidder). This helps us show sufficient profit for ALG. The analysis in [48] can be thought of as the special case when this map is the identity map.

4.2.2 Intuition

At first thought it seems very surprising that this algorithm would perform well in the presence of extra constraints. After all it seems to be ignoring the information provided by, say, the q-budgets for different q, while OPT may be using this information to its benefit. The reason why the algorithm works is that these particular extra constraints that we add turn out to be fortuitously geared to help the algorithm. The following example illustrates this in the context of the Greedy algorithm (recall that this has factor 1/2 in the standard model):

Consider the example shown in [36, 48] on which the greedy has a competitive ratio of 1/2 (for the standard bidding model). There are two bidders b_1 and b_2. Bidder b_1 is interested in keywords w_1 and w_2, whereas bidder b_2 is only interested in keyword w_2. Both have a budget of N, and a bid of 1. Now consider the sequence in which N queries of keyword w_2 come first and then N queries of keyword w_1 come next. OPT

will allocate all the w_2's to bidder b_2 and all the w_1's to bidder b_1, thus earning $2N$ units of money. Whereas greedy could assign all the w_2's to bidder b_1, and after that when the w_1's come in bidder b_1 cannot buy them, as he has exhausted his budget. So all the w_1's will go wasted, thus greedy will earn only N units of money, or about half of OPT.

The reason why greedy is factor 1/2 is that we allocate too many w_2's to bidder b_1. Now consider the following two cases in the q-budget version.

Case 1: b_1 has a $w_2 - budget$ constraint. Now, clearly there is a limit to which greedy could go bad as it cannot allocate *too many* w_2's to bidder b_1. So adding a $w_2 - budget$ to the standard model cannot make greedy worse.

Case 2: b_1 has a $w_1 - budget$ constraint. This case seems more dangerous, because greedy could do badly by allocating too many w_2's to b_1. But this is not the case, since greedy's allocation becomes bad only when OPT allocates too many w_1's to b_1. But a $w_1 - budget$ would not allow OPT to gain too much in this case.

Thus in one case, the q-budget prevents Greedy from taking a bad step, and in the other case it makes sure that what Greedy missed was not useful anyway. In fact we will show in section 4.3 that greedy still retains the factor 1/2 in these models. The above ideas, though applied to Greedy, also hold for MSVV.

4.3 The Decreasing Valuations Bidding Language

4.3.1 A Warm-up: Analysis of Greedy

Recall that Greedy was factor 1/2 in the standard model. Does the factor change in the model with decreasing bids? We answer in the negative – Greedy retains its factor of 1/2.

Theorem 25 *The competitive ratio of Greedy algorithm in the decreasing valuation bid model is 1/2.*

Proof: Let S_i be the set of queries assigned by OPT to bidder i. We will use $greedy(Q)$, $OPT(Q)$ to denote the money earned by the queries Q in Greedy and OPT respectively. We will use $greedy(i)$, $OPT(i)$ to denote the money spent by bidder i in *greedy* and OPT respectively. We will show that

$$\forall\ i: \quad greedy(i) + greedy(S_i) \geq OPT(i)$$

Summing over all i, we get:

$$\sum_i (greedy(i) + greedy(S_i)) \geq OPT$$

Now any query q can be counted atmost twice in the summation: $\sum_i (greedy(i) + greedy(S_i))$. Hence, $2 * greedy \geq OPT$, as required.

Now it remains to prove the equation above. If $greedy(i) \geq OPT(i)$, then we're done. So assume that $greedy(i) < OPT(i)$. This means that $greedy(i) < B_i$, so bidder i has remaining budget even at the end of Greedy. Now since the greedy allocates to the highest bidder, therefore all the queries of S_i which are not assigned by greedy to bidder i (call the set of such queries $S_i' \subseteq S_i$) must be making at least what bidder i is offering. Let S_i^w be the restriction of S_i' to keyword w. We only need to consider keywords w' s.t. the number of w' queries allocated by greedy to bidder i is less than that allocated by OPT to i. Since the bidding function is non-increasing, therefore all the queries in $S_i^{w'}$ are offered more money from bidder i in greedy than what OPT made by allocating $S_i^{w'}$ to i. Therefore collectively these queries must be making a profit of atleast $(OPT(i, w') - greedy(i, w'))$ in the greedy, where $OPT(i, w'), greedy(i, w')$ is the money spent by bidder i on keyword w' in OPT and greedy respectively. Summing over all keywords w, we get $greedy(S_i') \geq (OPT(i) - greedy(i))$.

Therefore, $greedy(S_i) \geq (OPT(i) - greedy(i))$. Hence, $\forall i, greedy(i) + greedy(S_i) \geq OPT(i)$. Note how we have crucially used the fact that the bid curves are decreasing.

□

4.3.2 Analysis of MSVV

In this section we will analyze the performance of the MSVV algorithm in the decreasing bids model. Let us recall the algorithm in the standard model. For clarification, we will use w to name a keyword, and q to name a query – a query q can be of type w.

The Algorithm. For the next query q (of type w) compute the *effective bid* of bidder i as: $b_{iw}\psi(y)$ where y is the fraction of budget spent by i, and $\psi(y) = 1 - e^{-(1-y)}$. Award q to the bidder with the highest effective bid.

In the decreasing bids model, the effective bid becomes:

$$f_{iw}(x)\psi(\text{fraction of budget spent by } i)$$

where x is the number of queries of keyword w already allocated to i.
We will prove the following theorem:

Theorem 26 *MSVV achieves a factor of* $1 - 1/e$.

We will follow the proof structure as in [48]. The crucial difference in the proof is in a careful charging via a well-chosen map between queries. We start with some preliminary notation.

Start by picking a large integer parameter k. Define the *type* of a bidder according to the fraction of budget spent by that bidder at the end of the algorithm BALANCE: say that the bidder is of type j if the fraction of his budget spent at the end of the algorithm lies in the range $((j-1)/k, j/k]$. Slab i is the portion of money $[(i-1)/k, i/k]$ of all the bidders.

Also, let α_j denote the number of bidders of type j. Let β_i denote the total money spent by the bidders from slab i in the run of the algorithm. It is easy to see that

$\beta_1 = N/k$, and

$$\forall\ 2 \leq i \leq k, \quad \beta_i = N/k - (\alpha_1 + \ldots + \alpha_{i-1})/k \tag{10}$$

Let $ALG(q)$ ($OPT(q)$) denote the revenue earned by the algorithm (OPT) for query q. Say that a query q is of *type* i if OPT assigns it to a bidder of type i, and say that q lies in *slab* i if the algorithm pays for it from slab i.

This concludes the notation from [48]. Fix a keyword w, and let Q_w be the set of all queries of keyword w, and let Q_w^{OPT} be the set of queries of keyword w assigned by OPT to all the bidders of type *strictly less* than k (these are the bidders who haven't spent all their money). We will drop the subscript w when it is clear by context. We will use subscript i to denote the restriction of any variable to bidder i.

Let q_{ij} denote the j^{th} query of keyword w allocated to the bidder i. Since $f_{i,w}$ is a decreasing function, therefore $f_{i,w}(q_{ij}) \leq f_{i,w}(q_{i(j-1)})$. Let q_{ij}^O, q_{ij}^A denote the allocation by OPT and ALG respectively. The inequality above holds for both these allocations.

Lemma 27 *For each keyword w, there exists a injective map $\sigma : Q_w^{OPT} \rightarrow Q_w$ s.t.* $\forall q \in Q_w^{OPT}$,

$OPT(q)\psi(type(q)) \leq ALG(\sigma(q))\psi(slab(\sigma(q)))$

Proof: To prove the lemma, we will construct the mapping for every query in Q_w^{OPT} such that the lemma holds (we will drop the w subscript henceforth).We do this in two phases.

Phase 1: For each bidder i, define

$x_i = min(|Q_i^{OPT}|, |Q_i^{ALG}|)$. We define a mapping for the first x_i queries in the OPT allocation:

$$\forall\ t \in [1, x_i] : \text{Define} \quad \sigma(q_{it}^O) := q_{it}^A$$

Therefore,

$$ALG(\sigma(q_{it}^O)) = ALG(q_{it}^A) = OPT(q_{it}^O)$$

Since, by definition, $type(q_{it}^O) \geq slab(q_{it}^A)$, therefore,

$$\psi(type(q_{it}^O)) \leq \psi(slab(q_{it}^A))$$

$$OPT(q_{it}^O)\psi(type(q_{it}^O)) \leq ALG(\sigma(q_{it}^O))\psi(slab(\sigma(q_{it}^O)))$$

Hence for the queries q mapped in first phase, we get:

$$OPT(q)\psi(type(q)) \ \leq \ ALG(\sigma(q))\psi(slab(\sigma(q)))$$

Phase 2: Lets call the queries from Q^{OPT} which were mapped in phase 1 to be $Q1^{OPT}$. Define $Q2^{OPT} = Q^{OPT} - Q1^{OPT}$. Similarly call the queries of Q to which queries from Q^{OPT} got mapped in the phase 1 to be $Q1$. Define $Q2 = Q - Q1$. Now look at a query $q \in Q2^{OPT}$ s.t. $OPT(q)\psi(type(q))$ is maximum over all the queries in $Q2^{OPT}$. Now look at any query $q' \in Q2$. We shall show that $OPT(q)\psi(type(q)) \leq ALG(q')\psi(slab(q'))$. Lets say OPT assigned query q to the bidder i_q. Since in the first phase we mapped $min(|Q_{i_q}^{OPT}|, |Q_{i_q}^{ALG}|)$ queries, and $q \in Q_{i_q}^{OPT}$ was not mapped, hence x_{i_q} must be equal to $|Q_i^{ALG}|$. Now, since the bidding function is decreasing, therefore wlg we can assume that $q =: q_{i_q(x_{i_q}+1)}^O$. When ALG was allocating q', then the bid of bidder i_q was $OPT(q)$. Hence by the allocation policy of the algorithm, $OPT(q)\psi(type(q)) \leq OPT(q)\psi(slab(q)) \leq ALG(q')\psi(slab(q'))$.

Therefore, in particular,

$$\forall q \in Q2^{OPT}, OPT(q)\psi(type(q)) \leq ALG(\sigma(q))\psi(slab(\sigma(q)))$$

Hence the lemma holds.

□

Let α_i^w be the money obtained by OPT from bidders of type i with keyword w only. Similarly let β_i^w be the portion of money obtained by ALG in slab i from keyword w only. Now we will aggregate the above result to prove the following lemma.

Lemma 28 $\sum_{i=1}^{k-1} \psi(i)(\alpha_i^w - \beta_i^w) \leq 0$

Proof: Let Q_w be the set of queries of keyword w. From lemma 27, we get:

$$\sum_{q \in Q_w : type(q) \leq k-1} [OPT(q)\psi(type(q)) - ALG(\sigma(q))\psi(slab(\sigma(q)))] \leq 0$$

$$\sum_{q \in Q_w : type(q) \leq k-1} OPT(q)\psi(type(q)) \leq \sum_{q \in Q_w : type(q) \leq k-1} ALG(\sigma(q))\psi(slab(\sigma(q)))$$

$$\sum_{q \in Q_w : type(q) \leq k-1} OPT(q)\psi(type(q)) \leq \sum_{q \in Q_w : type(q) \leq k-1} ALG(q)\psi(slab(q))$$

Now notice that in the MSVV algorithm, whenever a bidder hits k^{th} slab, algorithm stops allocating any more queries to him, hence

$$\sum_{q \in Q_w} ALG(q)\psi(slab(q)) = \sum_{i=1}^{k-1} \sum_{q \in Q_w : slab(q) = i} ALG(q)\psi(i)$$

$$= \sum_{i=1}^{k-1} \psi(i)\beta_i^w$$

.

Also,

$$\sum_{q \in Q_w : type(q) \leq k-1} OPT(q)\psi(type(q)) = \sum_{i=1}^{k-1} \sum_{q \in Q_w : type(q) = i} OPT(q)\psi(i)$$

$$= \sum_{i=1}^{k-1} \psi(i)\alpha_i^w$$

.

□

Now summing over all the keywords w, we get

Corollary 29 $\sum_{i=1}^{k-1} \psi(i)(\alpha_i - \beta_i) \leq 0$

Now, the calculations follow as in [48]: Using Corollary 29, Equation 10 and the definition of the trade-off function ψ, we get that the loss of the algorithm is at most OPT/e, hence proving the theorem.

4.4 Discussions

4.4.1 Towards more Expressive Models

We show that providing even slightly more (non-trivial) expressiveness leads to computational issues. Consider the case of *Group Budgets*: Instead of restricting to local budget constraints on a single keyword, the bidders are allowed to set a local budget on a group of keywords.

Suppose that the set of keywords is $\{w_0, w_1, .., w_k\}$. Let c_w represent the number of queries of keyword w. Consider the following instance: There is a single bidder and his bid on all the keywords is one dollar. His budget is, lets say, $t * k$, and has following k constraints on group of keywords:

$\forall i \in [1, k]$, $c_{w_0} + c_{w_i} \leq t$

Now consider the two sequence of queries ($w_0 \;\; t \;\; times$) and ($w_0 \;\; t \;\; times$, $w_1 \;\; t \;\; times, \cdots, w_k \;\; t \;\; times$). Its easy to see that: No randomized algorithm can do better than 1/2 on both the sequences. Also Greedy has a factor $1/k$ on the second sequence. Thus the extension to group budgets loses the computational possibilities available in our decreasing bids model. We believe that our bidding model achieves the correct trade-off between simplicity, expressivity and computational complexity.

4.4.2 Beyond the factor $1 - 1/e$

We now show how tightening Corollary 29 helps in getting bounds better than 1-1/e. Later we will try to see the conditions which tighten the Corollary 29.

Suppose we had that $\sum_{i=1}^{k-1} \psi(i)(\alpha_i - \beta_i) \leq -x$. This $(-x)$ goes directly to the objective function of the dual LP considered in the analysis of MSVV (see[48] for details). Since the objective function of the dual represents the maximum loss of the algorithm as compared to OPT, hence the total loss becomes $\frac{OPT}{e} - x$, and the competitive ratio of the algorithm will be $(1 - \frac{1}{e} + \frac{x}{OPT})$.

What are the cases when the x value is substantial? We believe that one case is when the bid curves decrease rapidly. The intuition is that if for a bidder i and keyword w, the bad case that the algorithm allocates less queries of type w to i than OPT does, is actually a good case. This is so because OPT derives much lesser profit for the extra queries (since they are farther in the bid-curve), while ALG allocates these elsewhere, more profitably. Characterizing this gain over $1 - 1/e$ in terms of input parameters (such as the derivative of the bid-curves) remains an open question.

CHAPTER V

MULTI-AGENT COVERING PROBLEMS

In this chapter, we introduce and study combinatorial problems with multi-agent submodular cost functions. We look at some well known covering problems in this setting and establish matching lower and upper bounds for the approximability of these problems.

A multitude of fundamental computational problems with real-world applications can be cast in the following framework: We are given a set X of elements, a collection C of subsets of X (i.e. $C \subseteq 2^X$) and a cost function f over the subsets of X. The collection C is typically specified via a combinatorial structure like a matroid or a graph property (for instance, the set of all spanning trees in a graph). The objective is to select a set $S \in C$ that minimizes $f(S)$.

Motivated by the considerations mentioned in the introduction to the thesis, we define the following class of *combinatorial problems with multi-agent submodular cost functions* (MSCP) problems - We are given a set of elements X and a collection $C \subseteq 2^X$. We are also given k agents, where each agent i specifies a normalized monotone submodular cost function $f_i : 2^X \rightarrow R^+$. The goal is to find a set $S \in C$ and a partition $S_1, ..., S_k$ of S such that $\sum_i f_i(S_i)$ is minimized.

There has been some work along these lines [10, 22, 61, 64, 27], but to the best of our knowledge, none of them has studied multi-agent submodular functions over a truly combinatorial structure. For instance, the work of [10] studies submodular function maximization over matroid constraints only in the single agent setting, the work of [61, 64] considers the Multi-Agent submodular functions setting but the combinatorial structure (or the collection C) used in their problem is either the set

of all subsets or the whole set itself.

Submodular functions form a rich class and capture the natural properties of economies of scale or the law of diminishing returns . A function $f : 2^X \rightarrow R^+$ is said to be submodular iff for any two sets S and $T \subseteq X$, $f(S)+f(T) \geq f(S \cup T)+f(S \cap T)$. Function f is said to be monotone if $f(S) \leq f(T)$ for any $S \subseteq T$, and normalized if $f(\emptyset) = 0$. Note that linear functions, i.e. functions of the form $f(S) = \sum_{i \in S} a_i$, are a special and classically studied subcase of submodular functions, Since a submodular function is defined over an exponentially large domain, we will work with the *value oracle* model in which an oracle will return the value of $f(S)$, when queried with the set $S \subseteq X$.

By fixing the collection C to any particular combinatorial structure, one can define a subclass of the problems of interest. In this contribution, we study the following fundamental subclass of problems in MSCP :

- **Combinatorial Reverse Auction (CRA)**: We are given a set X of elements and the collection C consists of only the set X, i.e. in the required solution all the elements must be covered. This models the situation where a set of jobs needs to be assigned to multiple workers.

- **Submodular Vertex Cover (MS-VC)**: We are given an undirected graph $G(V, E)$. Element set X is the same as the set of vertices V and the collection C consists of all the vertex covers of the graph. Recall that a set $S \subseteq V$ is a vertex cover if every $e \in E$ is incident on a vertex in S.

- **Submodular Minimum Perfect Matchings (MS-MPM)**: In this setting, we have a undirected graph $G = (V, E)$ with cost functions over E. G contains at least one perfect matching. Element set X is the set of all edges, and the collection C is defined as the set of all perfect matchings of G. Recall that a set $M \subseteq E$ is a perfect matching of G if exactly one edge in M is incident on

every vertex.

- **Submodular Minimum Spanning Tree (MS-MST)**: We are given a connected undirected graph $G = (V, E)$ with cost functions over E. Element set X is the set of all edges, and the collection C is the set of spanning trees of G. Recall that a spanning tree is a minimal connected subgraph of G.

For each of the above problems, we study both the single agent and the multi-agent setting.

5.0.3 Motivation and Applications

In this contribution, we propose to extend the algorithmic study of covering type problems to the more general setting of submodularity and multiple agents. From a practical viewpoint, each of the problem we study is meaningful in its own right. Combinatorial reverse auctions are used by buyers to obtain required goods or services from various sellers, for instance the procurement of components required to build high-end goods like automobiles or computers. Spanning trees are used in network design problems, and it is reasonable to assume that different agents could have different submodular cost functions depending on the set of edges they can construct cheaply. Similarly one can motivate rest of the problems as they are used in many settings, especially those related to algorithmic game theory.

From a theoretical perspective, one would like to extend the tools and techniques developed for combinatorial problems with linear cost functions to as general a setting as possible. Submodular functions are a natural generalization where one would expect to be able to extend the techniques. Despite the significant recent progress on some of the fundamental problems in this area [10, 22, 61, 64, 27], the algorithmic theory for combinatorial problems with submodular cost functions is not substantially developed yet. Many more basic questions remain to be identified and solved, which could form the basis of tools and techniques for solving more complex problems.

5.0.4 Our Results

We give an approximation algorithm and a matching information theoretic lower bound for each of the subclass of problems that we mentioned earlier. We present these results in the table below.

Table 2: Results

	Single-agent		Multi-agent	
	Lower bound	Upper bound	L. bound	U. bound
Reverse Auction	1	1	$\Omega(\log n)$	$O(\log n)$ [30]
Vertex Cover	$2-\epsilon$	2	$\Omega(\log n)$	$O(\log n)$
Perfect Matching	$\Omega(n)$	$O(n)$	$\Omega(n)$	$O(n)$
Spanning Tree	$\Omega(n)$	$O(n)$	$\Omega(n)$	$O(n)$

Note that the minimum perfect matching and minimum spanning tree problems, which are polynomial time solvable with linear cost functions, become very hard with submodular cost functions.

We would like to draw attention to our hardness result for the Vertex Cover problem in the single agent case. In the classical vertex cover problem, the best known approximation factor is 2, and the best known hardness of approximation is 1.3606 (assuming $P \neq NP$) [18]. Khot et al. [41] showed that achieving a factor of $2-\epsilon$ 'might be' hard by showing hardness result based on UGC conjecture [39]. Our results for Vertex Cover in the single agent case implies that, if the cost function over the set of vertices is submodular, then the optimal approximation factor is indeed 2.

Our hardness results use information theoretic arguments and follow the framework explained in Section 5.1, along with special manipulations specific to each problem. Our algorithms are based on LP rounding or greedy methods.

We would like to point out that our results for perfect matchings and spanning trees extend to the class of subadditive cost functions, and to related combinatorial structures such as steiner trees.

5.0.5 Related Work

Submodular functions have been of great interest in Optimization in the past. The most fundamental of them is, perhaps, the non-monotone submodular function minimization problem. A sequence of works in this direction [55, 33, 31, 53, 32, 34] has resulted in fast strongly polynomial time combinatorial algorithms. Another related work is that of non-monotone submodular function maximization [22]. Both these algorithms are sometimes used as a subroutine in solving the configuration LPs corresponding to some other submodular combinatorial optimization problem.

Another body of work in optimization over submodular functions deals with welfare maximization [10, 64, 20, 40]. Calinescu et al. [10] studied submodular function maximization subject to matroid constraints. They showed that their problem contains as a subcase many other allocation problems, thus giving a unified framework for studying such problems. Matching information theoretic lower bounds were established in [49].

Very recently, Svitkina and Fleischer [61] studied submodular objective function for problems like Sparsest cut, load balancing, and knapsack. They gave $O(\sqrt{\frac{n}{\log n}})$ upper and lower bounds for all these problems, showing that all these problems become much more hard in the submodular setting. Goemans et al. [27] gave an algorithm for constructing explicit approximate submodular functions by querying polynomial number of times to the original submodular function. Some other related work in optimization that uses submodular functions include [56, 30, 62, 60, 65]. For the Combinatorial Reverse Auction problem, [30] noted that the generalization of greedy algorithm for Set Cover problem gives a $\log(n)$ approximation ratio. For the lower bound of the Combinatorial Reverse Auction, it turns out that we only rediscovered the factor, as we later found out that Nisan [52] has already proved similar result, though using a slightly different technique.

Recall that the problems we study in this contribution are very well studied in

the linear cost setting. Min cost perfect matching and spanning tree can be solved exactly in polynomial time. Algorithm for vertex cover with factor 2 for weighted graphs was first given by [5]. The best known hardness of approximation for Vertex Cover is 1.3606 (assuming $P \neq NP$) [18]. Using UGC conjecture [39], Khot and Regev [41] showed that achieving a factor of $2 - \epsilon$ is hard. Combinatorial reverse auction generalizes the set cover problem. A simple greedy algorithm gives a factor $\log n$ for the set cover, and Feige [21] showed that this factor is essentially the best possible by showing a matching hardness of approximation result, under the assumption that NP complete problems does not exhibit quasi-polynomial time algorithms.

5.1 Preliminaries: Information Theoretic Lower Bounds

In this contribution we establish the complexity of some multi-agent submodular combinatorial optimization problems. A problem is said to have information theoretic lower bound of α if any randomized algorithm that approximates the optimum to a factor α with high probability requires superpolynomial number of queries to the value oracle.

By Yao's principle, it suffices to establish the lower bounds for deterministic algorithms acting on an input which is picked randomly from some fixed distribution. To show these approximation gaps, we follow the general framework which was also used in [61, 27, 22]. We will outline this framework in the single agent setting.

The idea is first to choose a problem instance which has a suitably large collection set $C \subseteq 2^X$ of interest. For example, for the spanning tree problem, we choose a graph that has exponentially many spanning trees. Then we design two submodular cost functions f and g. Typically, g is deterministically picked, whereas f is chosen from a distribution. The choice of f and g relies on the following two properties: a) f and g must be 'hard to distinguish' in the sense that they return the same value on almost all queries and b) The optimum values of f and g over C must differ by

a large factor. Intuitively this amounts to 'hiding' a particular set $Q \in C$ in f by setting $f(Q)$ to a low value. The set Q is chosen from a distribution over C. Since C is designed to be extremely large, this leaves an exponentially small probability of an arbitrary query S made by an algorithm differentiating f from g. By the union bound and a computation path argument [61, 22], an algorithm making polynomially many number of queries cannot distinguish between f and g. Combining this with the gap in the optima of f and g, one proves the lower bound.

We note an important observation from the above discussion, which we will use in our proofs of lower bounds:

Observation 5.1.1 *To prove an information theoretic lower bound using two submodular functions f and g with a gap in their optimum values, it suffices to prove that $Pr[\ f(Q) \neq\ g(Q)\ ]$ is exponentially small over the random choice of $Q \subseteq X$.*

The above framework can be extended to the multi-agent setting by choosing two sets of submodular functions.

5.2 Combinatorial Reverse Auction

In this problem we are given a set J, of n elements and m agents. For each agent i we have a normalized monotone submodular cost function $f_i : 2^J \rightarrow \mathbb{R}^+$. We wish to partition the elements among the agents to minimize the total cost. We prove a $\Omega(\log n)$ information theoretic lower bound on the hardness of approximation and provide an algorithm that matches this bound. We also prove the same algorithm to be m-approximate.

5.2.1 Proof of hardness

In this section, we derive an $\Omega(\log n)$ lower bound on the hardness of approximation for the CRA problem. The idea is to construct a deterministic instance and a random instance of the CRA problem so that the optimal solution of these two instances

differ by a factor of $\Omega(\log n)$, and then show that with high probability, a deterministic algorithm which uses only polynomially many value queries can not distinguish between these two instances.

Consider the following deterministic instance of CRA: There are m agents and a set J of $n = m(m+1)^2/4$ elements. The elements are equally partitioned into m blocks $J_1, J_2, ..., J_m$. We will choose m such that $m = 2^d - 1$, for some d. Now each number i between 1 and m can be represented as a vector $\mathbf{a}_i$ in $GF[2]^d$. Let $G_i = \bigcup_{1 \leq k \leq m,\ \mathbf{a}_i \cdot \mathbf{a}_k = 1} J_k$. For each i, $1 \leq i \leq m$, agent i is only interested in elements in G_i. It can be shown that $|\{\mathbf{a}_k : \mathbf{a}_i \cdot \mathbf{a}_k = 1\}| = (m+1)/2$. Thus each agent is interested in only $(m+1)/2$ blocks of elements and for each block there are $(m+1)/2$ agents who are interested in it. Now, we define the cost function $f_i : 2^J \longrightarrow \mathbb{R}^+$ as follows:

$$f_i(S) = \begin{cases} \min\{|S|, (m+1)^2/4\}, & \text{if } S \subseteq G_i \\ \infty, & \text{otherwise} \end{cases}$$

Let us analyze the optimal cost of this instance. We say that an agent is *marked* if the total size of elements assigned to his is at least $(m+1)^2/4$. Among all the optimal solutions, let OPT be the one that maximizes the number of marked agents. We claim that at least d agents are marked in OPT. Suppose not, then without loss of generality, we may assume $M = \{1, 2, ..., t\}$ to be the set of marked agents and $t < d$. The system of linear equations $\mathbf{a}_i \cdot \mathbf{x} = 0, \forall 1 \leq i \leq t$ has at least one solution $\mathbf{x}^* \in GF[2]^d$, since number of equations is less than the number of variables. Let k be the number between 1 and m corresponding to the vector $\mathbf{x}^*$. This implies that no agent in M is interested in block J_k. Let $A_k = \{i_1, i_2, ..., i_w\}$ be the set of agents who are assigned some elements in J_k. Then $A_k \cap M = \emptyset$. Therefore, we can mark one more agent by transferring the elements in J_k from agents $i_2, i_2, ..., i_w$ to agent i_1 without changing the cost of the new solution. This is a contradiction because of the choice of OPT. Hence, the optimal cost of this instance is at least $(m+1)^2 d/4$.

For the random instance, we have the same sets of agents and elements. Also, each agent is interested in the same set of elements. However, the cost function for each agent is defined by a probability distribution on the set assigned to her. Next we describe our construction of the random cost functions explicitly.

For each element, assign it uniformly at random to one of the agents who is interested in it. Let S_i be the set of elements which agent i gets. Clearly $(S_1, S_2, ..., S_m)$ forms a partition of the element set J. We define the cost function $g_i : 2^J \longrightarrow \mathbb{R}^+$, for agent i as follows:

$$g_i(S) = \begin{cases} \min\{\ |S \cap \overline{S_i}| + \min\{|S \cap S_i|,\ (1+\delta)(m+1)/2\ \},\ (m+1)^2/4\}, & \text{if } S \subseteq G_i \\ \infty, & \text{otherwise} \end{cases}$$

where $\delta > 0$ is a fixed constant.

Now we show that with high probability, a deterministic algorithm using only polynomially many value queries can not distinguish between $\mathbf{f} = (f_1, f_2, ..., f_m)$ and $\mathbf{g} = (g_1, g_2, ..., g_m)$. We prove the following lemma.

Lemma 30 *For any subset S of elements and any i, $1 \leq i \leq m$, $Pr[f_i(S) \neq g_i(S)] = e^{-\Omega(m)}$.*

Proof:

Suppose S is a subset of elements and $1 \leq i \leq m$. By our construction, $g_i(S) \leq f_i(S)$. Therefore $Pr[f_i(S) \neq g_i(S)] = Pr[g_i(S) < f_i(S)]$.

First of all, we claim that the above probability is maximized when $S \subseteq G_i$ and $|S| = (m+1)^2/4$. For this, if $S \not\subseteq G_i$, then $f_i(S) = g_i(S) = \infty$ hence $Pr[f_i(S) < g_i(S)] = 0$. Now suppose $S \subseteq G_i$ and $|S| \geq (m+1)^2/4$. Then $f_i(S) = (m+1)^2/4$. Therefore,

$$Pr[g_i(S) < f_i(S)] = Pr[|S \cap \overline{S_i}| + \min\{|S \cap S_i|,\ (1+\delta)(m+1)/2\ \} < (m+1)^2/4]$$

This probability can only increase when we remove elements from S. For the case when

$|S| \leq (m+1)^2/4$, we get

$$\begin{aligned} Pr[g_i(S) < f_i(S)] &= Pr[|S \cap \overline{S_i}| + \min\{|S \cap S_i|,\ (1+\delta)(m+1)/2\ \} < |S|] \\ &= Pr[\min\{|S \cap S_i|,\ (1+\delta)(m+1)/2\ \} < |S \cap S_i|] \\ &= Pr[|S \cap S_i| > (1+\delta)(m+1)/2] \end{aligned}$$

Thus, this probability can only increase when more elements are added to S. Hence under the condition $S \subseteq G_i, |S| \leq (m+1)^2/4$, the probability is also maximized when $|S| = (m+1)^2/4$.

Now we assume $S \subseteq G_i$ and $|S| = (m+1)^2/4$. In this case, $Pr[g_i(S) < f_i(S)] = Pr[|S \cap S_i| > (1+\delta)(m+1)/2]$, which by a standard Chernoff bound arguments, can be shown to be bounded by $e^{-\Omega(m)}$. $\square$

If we define $\mathbf{f}(S) = (f_1(S), ..., f_m(S))$ and $\mathbf{g}(S) = (g_1(S), ..., g_m(S))$, then by a simple union bound, as a corollary of the lemma, we have $Pr[\mathbf{f}(S) \neq \mathbf{g}(S)] = poly(m)e^{-\Omega(m)}$. Now suppose $\mathcal{A}$ is a deterministic algorithm which makes polynomially many queries to the value oracle. Then by the union bound, with probability at most $poly(m) \cdot e^{-\Omega(m)}$, $\mathcal{A}$ can distinguish between $\mathbf{f}$ and $\mathbf{g}$. Notice that for the cost function $\mathbf{g} = (g_1, ..., g_m)$, the optimal solution is at most $(1+\delta)m(m+1)/2$ achieved by assigning S_i to agent i. However, as we showed, the optimal solution for the cost function $\mathbf{f} = (f_1, f_2, ..., f_m)$ has cost at least $d(m+1)^2/4$, thus with high probability, $\mathcal{A}$ can not approximate a **CRA** instance within factor $\frac{(m+1)^2 d/4}{(1+\delta)m(m+1)/2} \simeq d = c \log n$ for some $c < 1$.

At last, by Yao's principle, we have the following:

Theorem 31 *A randomized approximation algorithm for the **CRA** problem within factor $c \log n$ for some $c < 1$ needs to make exponentially many value queries.*

5.2.2 A $\min(m, \log n)$ approximation algorithm for combinatorial reverse auction

A $\log n$ approximate algorithm for this problem appeared in [30]. In what follows we provide a $\min(m, \log n)$ approximation algorithm. Consider the following LP relaxation (LP1) and its dual (LP2).

$$\min \sum_{S \subseteq V} \sum_{i} x_{i,S} f_i(S) \qquad \text{(LP1)}$$

$$\sum_{S: u \in S} \sum_{i} x_{i,S} \geq 1 \qquad \forall u \in X$$

$$x_{i,S} \geq 0 \qquad \forall S \subseteq V,\ \forall i$$

$$\max \sum_{u \in X} y_u \qquad \text{(LP2)}$$

$$\sum_{u \in S} y_u \leq f_i(S) \qquad \forall S \subseteq V,\ \forall i$$

$$y_u \geq 0 \qquad \forall u \in X$$

In LP1, $x_{i,S}$ is used to represent the fraction of set S that is allocated to agent i. Since $f_i(S) - \sum_{u \in S} y_u$ is a submodular function, we can construct a separation oracle for the dual program using the submodular minimization algorithm as a subroutine. Thus we can solve LP1 and LP2 optimally. The following lemma describes the structure of an optimal solution to LP1.

Lemma 32 *There exists an optimal fractional solution to LP1 such that for every agent i the set $\mathcal{T}_i = \{\ S\ :\ x_{i,S} > 0\ \}$ forms a nested family.*

Proof:

Let x be any feasible solution to LP1. If $\mathcal{T}_i$ is not nested, then there exist $A, B \in \mathcal{T}_i$ such that neither A nor B is contained in the other. We may assume $x_{i,A} \geq x_{i,B}$. We will construct another feasible solution x' to LP1 as follows:

- $x'_{i,A \cup B} = x_{i,B}$
- $x'_{i,B} = 0$
- $x'_{i,S} = x_{i,S}$ for all $S \in X$ other than the above.
- $x'_{i,A} = x_{i,A} - x_{i,B}$
- $x'_{i,A \cap B} = x_{i,B}$ if $A \cap B \neq \emptyset$
- $x'_{j,S} = x_{j,S}\ \forall j \neq i$ and $\forall S \in X$

By submodularity, one can verify that the cost of the solution x' is at most the cost of x. If the set $\mathcal{T}'_i$ corresponding to x' is nested, we are done. Otherwise, we repeat the procedure for x'. The termination of the above procedure can be guaranteed by observing that the potential function $\sum_{S \in \mathcal{T}_i} |S|^2$ strictly increases and is polynomially bounded. □

Let $\overline{x}$ be an optimal solution of LP1 with cost W, satisfying the conditions in lemma 32 and $\mathcal{T}_i$ be the corresponding nested families of sets. Let $\mathcal{T} = \bigcup_i \mathcal{T}_i$. Let Y denote the set of uncovered elements in X. In each iteration pick the set $(i, S) \in \mathcal{T}$ minimizing $f_i(S)/|S \cap Y|$. Add S to the cover and assign it to agent i. Remove all the newly covered elements from Y. Repeat until all elements are covered. Since each $\mathcal{T}_i$ is a nested family, an agent can

drop all but the largest set assigned to her. Let (i, S) be the set covering an element u in the integral cover. Then we define $\alpha(u) = f_i(S)/|S \cap Y|$ to be the cost 'borne' by u. Note that $\sum_u \alpha(u)$ is exactly the cost of the integral cover.

Let $u \in X$ be the j'th element to be covered by this algorithm and let (i, S) be the set chosen to cover it. Suppose u was picked during the algorithm. Then since $\overline{x}$ is a fractional cover of Y, $f_i(S)/|S \cap Y| \leq W/|Y|$.

$$\alpha(u) \leq f_i(S)/|S \cap Y| \leq W/|Y| = W/(|X| - j + 1)$$

On the other hand, if u was not picked by the algorithm, then $\alpha(u) \leq W/(|X| - j' + 1) \leq W/(|X| - j + 1)$ for some $j' < j$.

Summing over all u, we conclude that the integral cover has cost at most $W \log n$. To prove that this algorithm is also m-approximate, observe that each set selected has cost at most W. Moreover, each agent is assigned at most one set in the final solution. This proves the claim.

5.3 Vertex Cover

In this section, we consider the submodular vertex cover problem. We first prove an information theoretic lower bound of $2 - \epsilon$ on the hardness of approximation in the single agent setting and provide an algorithm with approximation ratio of 2. We present a $2 \log n$ approximation algorithm for the multi-agent case.

5.3.1 Single Agent Case

We are given an undirected graph $G(V, E)$ and a normalized monotone submodular function $f : 2^V \longrightarrow \mathbb{R}$. We wish to find a vertex cover $U \subseteq V$ such that $f(U)$ is minimized.

Theorem 33 *For every fixed $\epsilon > 0$, any randomized algorithm for the submodular vertex cover problem with an approximation ratio of $2 - \epsilon$ needs exponentially many queries to the value oracle.*

Proof: Choose δ such that $2/(1 + \delta) = 2 - \epsilon$.

Consider a bipartite graph $G(A \cup B, E)$ such that both A and B are of size n. The edge set consists of n edges matching vertices in A to those in B. Let R be a random minimum cardinality vertex cover of this graph, which can be picked by choosing one endpoint of every edge uniformly at random.

Define the following two submodular cost functions.

$$\begin{aligned} f(S) &= min\left\{ |S \cap \overline{R}| \ + \ min\left\{ |S \cap R|, \ \frac{(1+\delta)n}{2} \right\}, \ n \right\} \\ g(S) &= min\{ |S|, \ n \} \end{aligned}$$

Note that the optimum value of the vertex cover for the function f is $\frac{(1+\delta)n}{2}$, and for g it is n. Thus if we can show that any randomized algorithm, cannot distinguish between f and g with high probability, then it will imply an inapproximability ratio of $2/(1+\delta)$ or $2-\epsilon$ for the submodular vertex cover problem.

From Observation 5.1.1 it suffices to show that for a deterministic query Q, $Pr[\ f(Q) \neq g(Q)\ ]$ is exponentially small, where the probability space is defined over the random choice of set R. Since $f(S) \leq g(S)$ for all $S \subseteq V$, $f(Q) \neq g(Q)$ implies $f(Q) < g(Q)$.

Let Q^* be the optimal query for which $Pr[\ f(Q) < g(Q)\ ]$ is maximized. We will show that $|Q^*| = n$. Suppose $|Q| \geq n$, then

$$Pr[\ f(Q) < g(Q)\ ] = Pr[\ |Q \cap \overline{R}| \ + \ min\{\ |Q \cap R|, \ (1+\delta)n/2\ \} < n\ ],$$

which increases as the size of Q is reduced. Thus the size of the optimal query in this case is n.

Now suppose $|Q| \leq n$. In this case, it is not difficult to see that $Pr[\ f(Q) < g(Q)\ ] = Pr[\ |Q \cap R| > (1+\delta)n/2\ ]$, which increases as $|Q|$ is raised. Therefore, the optimal query size is n.

For $|Q| = n$, $E[|Q \cap R|] = n/2$. Now it is not difficult to show using Chernoff like bounds that $Pr[\ f(Q) < g(Q)\ ] = Pr[\ |Q \cap R| > (1+\delta)n/2\ ] \leq e^{-n\delta^2/3}$. Hence, the probability that an arbitrary query Q can distinguish between f and g is exponentially small. $\square$

Theorem 34 *There exists an algorithm which finds a 2-approximate solution to the single agent vertex cover problem with submodular costs.*

Proof: We use the following LP relaxation LP3.

$$\begin{array}{llll} \min \sum_{S \subseteq V} x_S f(S) & (LP3) & \max \sum_{e \in E} y_e & (Dual) \\ \sum_{S:u \in S} x_S + \sum_{S:v \in S} x_S \geq 1 & \forall (u,v) \in E & \sum_{v \in S} \sum_{e \in \delta(v)} y_e \leq f(S) & \forall S \subseteq V \\ x_S \geq 0 & \forall S \subseteq V & y_e \geq 0 & \forall e \in E \end{array}$$

Here the variable x_S is an indicator variable for the set S of vertices. It is not difficult to see that the function $\sum_{v \in S} \sum_{e \in \delta(v)} y_e$ is a modular function. Thus $f(S) - \sum_{v \in S} \sum_{e \in \delta(v)} y_e$ is a submodular function, and we can use the submodular minimization algorithm as a subroutine to construct a separation oracle for the dual. This allows us to find an optimal fractional solution to the LP3 with value OPT. Let x^* be this solution. Output $Q = \left\{ u \in V : \sum_{S:u \in S} x^*_S \geq 1/2 \right\}$ as the vertex cover. Clearly, for any $(u,v) \in E$, either $\sum_{S:u \in S} x^*_S \geq 1/2$ or $\sum_{S:v \in S} x^*_S \geq 1/2$ must hold, thus Q is a valid vertex cover of G. Since $2x^*$ is a fractional cover of Q, submodularity implies that $f(Q) \leq 2 \sum_{S \subseteq V} x^*_S f(S) = 2 \cdot \text{OPT}$. □

5.3.2 Multi-agent Case

We are given an undirected graph $G(V, E)$ and a normalized monotone submodular function $f_i : 2^V \longrightarrow \mathbb{R}$ for each agent i. We wish to find a vertex cover $U \subseteq V$ such that $f(U)$ is minimized.

Theorem 35 *Any randomized algorithm for the multi-agent submodular vertex cover problem with an approximation ratio $c \log n$ for some constant $c < 1$ needs exponentially many queries to the value oracle.*

Proof:

We will sketch the idea behind the proof, which is based on the hardness result for CRA (Theorem 31). We will use a suitable instance graph such as the one used in the proof of Theorem 33, and fix a vertex cover Q. We will then assign a very high value to $f_i(S)$

for any S that contains a vertex from $V - Q$ and for all i. Essentially, the problem then reduces to that of assigning vertices in Q to various agents so as to minimize the total cost. Notice that this situation looks exactly like a CRA problem. Therefore, we can now build a hardness example of the kind constructed in the proof of theorem 31 treating Q as the ground set. □

2 log n-approximate algorithm: We begin by finding an optimal solution $\overline{x}$ using the LP relaxation LP5. Let $Q = \left\{ u \in V \ : \ \sum_{S:u \in S} \sum_i \overline{x}_{i,S} \geq 1/2 \right\}$ be a vertex cover. We will now round $2\overline{x}$ to find an integral cover of Q. Let W denote the total cost of the solution $2\overline{x}$.

$$\begin{aligned} \min \sum_{S \subseteq V} \sum_i x_{i,S} f_i(S) \qquad & (LP5) \\ \sum_{S:u \in S} \sum_i x_{i,S} + \sum_{S:v \in S} \sum_i x_{i,S} \geq 1 \qquad & \forall (u,v) \in E \\ x_{i,S} \geq 0 \qquad & \forall S \subseteq V, \ \forall i \end{aligned}$$

At any step of the algorithm let R contain the uncovered elements in Q. For any fractional cover x of R, define $\alpha_x(u) = \sum_{S:u \in S} \sum_i \frac{x_{i,S} f_i(S)}{|S \cap R|}$. Note that $\sum_u \alpha_x(u) = \sum_{i,S} x_{i,s} f_i(S)$. Pick $u \in Q$ that minimizes $\alpha_{2\overline{x}}(u)$. Among the sets containing u, choose a set (i, S) randomly with probability proportional to $x_{i,S}$. Remove all the newly covered elements from R and iterate until all the elements in Q are covered. Let y denote this integral cover.

Let $u_1, u_2, ...$ be the order in which the vertices in Q get covered. We claim that $E[\alpha_y(u_j)] \leq W/(|Q| - j + 1)$. Suppose u_j was picked during the algorithm. Then, $E[\alpha_y(u_j)] \leq \alpha_{2\overline{x}}(u_j)$. Since $2\overline{x}$ covers the remaining $|Q| - j + 1$ elements in Q, $\alpha_{2\overline{x}}(u_j) \leq W/(|Q| - j + 1)$. On the other hand, if u_j was not picked during the algorithm, then

$$E[\alpha_y(u_j)] \ = \ \alpha_{2\overline{x}}(u'_j) \ \leq \ W/(|Q| - j' + 1) \ \leq \ W/(|Q| - j + 1)$$

for some $j' < j$. Summing over j, we have

$$\sum_{i,S} y_{i,S} f_i(S) \ = \ \sum_{u \in Q} \alpha_y(u) \ \leq \ \sum_{u \in Q} \alpha_{2\overline{x}}(u) \ \leq \ W \log n \ \leq \ \text{OPT} \cdot 2 \log n$$

This algorithm can be derandomized using standard techniques.

5.4 Perfect Matching

In this section, we consider the multi-agent submodular minimum perfect matching problem. In this problem we are given a bipartite graph $G(V, E)$ containing at least one perfect matching and a normalized monotone submodular function $f : 2^E \longrightarrow R^+$. We wish to find a perfect matching in G of minimum value. We prove an information theoretic lower bound of $\Omega(n)$ on the hardness of approximation for this problem and give an algorithm matching this bound.

As in previous sections, we proceed by designing two submodular functions that are hard to distinguish in polynomially many queries but have different optimal values. In the general framework outlined in section 5.1, this is accomplished by 'hiding' a random element of the target collection C in one of the functions. In this case, C is the set of all perfect matchings. Choosing a random matching from C however does not serve our purpose. For a fixed pair of edges, the events that these edges belong to the random matching are not independent precluding the use of Chernoff bounds. We address this problem using the following result on random graphs [8]:

Lemma 36 *Let $G(n, n, p)$ be a random bipartite graph on $2n$ vertices such that each edge is present independently with probability p. Then*

$$Pr[\ G(n, n, p)\ \textit{contains no perfect matching}\ ] = O(ne^{-np})$$

Essentially, we hide a collection of randomly and independently chosen edges that contains a perfect matching with high probability. The above lemma extends to many linear properties of random graphs, such as the spanning trees studied in the next section, and allows us to extend our results to those properties.

Theorem 37 *For every fixed $\epsilon, \delta > 0$, any randomized approximation algorithm for the submodular minimum perfect matching problem with factor $n^{1-3\epsilon}/(1+\delta)$ needs exponentially many queries.*

Proof: Consider the graph G which is a union of n^ϵ different copies of the complete bipartite graph $K_{n^{1-\epsilon},n^{1-\epsilon}}$. Let G_i be the i^{th} copy. In each copy G_i, we pick a random subset R_i

of edges by picking each edge independently with probability $p = 1/n^{1-2\epsilon}$. Let $R = \bigcup R_i$. By applying lemma 36 followed by the union bound, R contains a perfect matching with probability $1 - O(\text{poly}(n)e^{-n^\epsilon})$.

Define the following two submodular cost functions $f, g : 2^E \longrightarrow \mathbb{R}^+$:

$$\begin{aligned} f(Q) &= \min\{\ |Q \cap \overline{R}|\ +\ \min\{\ |Q \cap R|,\ (1+\delta)n^{3\epsilon}\ \}\,,\ n^{1+\epsilon}\ \} \\ g(Q) &= \min\{\ |Q|,\ n^{1+\epsilon}\ \} \end{aligned}$$

With probability $1 - O(\text{poly}(n)e^{-n^\epsilon})$, R contains a perfect matching and hence the minimum value of a perfect matching in f is $(1+\delta)n^{3\epsilon}$. Therefore the ratio of optima in g and f is at least $\frac{n^{1-3\epsilon}}{(1+\delta)}$ with high probability.

Now we look at the probability that the algorithm can not distinguish f and g. By observation 5.1.1 it suffices to prove that $Pr[f(Q) \neq g(Q)]$ is exponentially small for an arbitrary query Q. It's easy to see that $f(S) \leq g(S)$, thus these two functions differ on Q if and only if $f(Q) < g(Q)$.

Making arguments analogous to the proof of theorem 33, $Pr[\ f(Q) < g(Q)\ ]$ is maximized when $|Q| = n^{1+\epsilon}$. Therefore,

$$Pr[\ f(Q) < g(Q)\ ] = Pr[\ |Q \cap R| > (1+\delta)n^{3\epsilon}\ ]$$

Since $E[|Q \cap R|] = n^{3\epsilon}$ and we pick edges independently, we can apply Chernoff bounds to conclude that this probability is $O(e^{-n^{2\epsilon}\delta^2})$. This proves the theorem. □

Factor n approximation algorithm for MS-MPM: We are given a graph $G(V, E)$ and submodular cost functions f_i for each agent. Define a new weight function w over E as $w_e = \min_i f_i(\{e\})$. Run the minimum weight perfect matching algorithm on G with weight function $w(Z) = \sum_{e \in Z} w_e$ for all $Z \subseteq E$ to get a perfect matching M. Assign each edge $e \in M$ to the agent i minimizing $f_i(\{e\})$. Let the cost of this solution be W.

Claim 38 *M is an n-approximate solution to MS-MPM.*

By submodularity we have $W \leq w(M)$. Let M_0 be an optimal solution of MS-MPM having value OPT. Since M is a minimum weight matching under w, $w(M) \leq w(M_0)$. By

submodularity of the cost functions, $w(M_0) \leq n \cdot w_e$. By monotonicity we have $w_e \leq OPT$. Therefore,

$$W \ \leq \ w(M) \ \leq \ w(M_0) \ \leq \ n \cdot w_e \ \leq \ n \cdot OPT$$

This proves the claim.

5.5 Spanning Tree

In this section, we consider the multi-agent submodular minimum spanning tree problem. We are given a connected graph $G(V, E)$ and a normalized monotone submodular function $f : 2^E \longrightarrow R^+$. We want to find a spanning tree in G of minimum value. We prove an information theoretic lower bound of $\Omega(n)$ on the hardness of approximationand also present an algorithm matching this bound.

To prove the lower bound we will provide two submodular functions that can not be distinguished in polynomially many queries and have widely differing optimal values. As in Section 5.4, we will use the following lemma [8] in the proof.

Lemma 39 *Let $G(n, p)$ be a random graph on n vertices such that each edge is present independently with probability p. Then*

$$Pr[\ G(n,p) \ \textit{is disconnected}\ ] \leq n(1-p)^n.$$

Theorem 40 *For every fixed $\epsilon, \delta > 0$, any randomized approximation algorithm for the submodular minimum spanning problem on a with factor $n^{1-3\epsilon}/(1+\delta)$ needs exponentially many queries. Also there exists an algorithm that approximately finds an n-approximate spanning tree in polynomially many queries.*

Proof: For $1 \leq i \leq n^{\epsilon}$, let G_i be a clique on $n^{1-\epsilon}$ vertices. Suppose there exists a vertex v such that $V(G_i) \cap V(G_j) = \{v\}$ for any $i \neq j$. Let $G = \bigcup G_i$. We choose a random subset of edges R_i, in each copy G_i, by picking each edge independently with probability $p = 1/n^{(1-2\epsilon)}$. Let $R = \bigcup R_i$. By applying Lemma 39 followed by the union bound, R is connected with probability at least $1 - ne^{-n^{2\epsilon}}$.

Define the following two submodular cost functions $f, g : 2^E \longrightarrow \mathbb{R}^+$:

$$\begin{aligned} f(Q) &= \min\{\ |Q \cap \overline{R}|\ +\ \min\{\ |Q \cap R|,\ (1+\delta)n^{3\epsilon}\ \}\ ,\ n^{1+\epsilon}\ \} \\ g(Q) &= \min\{\ |Q|,\ n^{1+\epsilon}\ \} \end{aligned}$$

With probability $1 - O(\text{poly}(n)e^{-n^\epsilon})$, R is connected and hence, the value of the optimal spanning tree in f is $(1+\delta)n^{3\epsilon}$. Therefore, the ratio of optimal solution values in g and f is at least $\frac{n^{1-3\epsilon}}{(1+\delta)}$ with high probability.

Making arguments similar to the proof of Theorem 37, we conclude that $Pr[\ f(Q) < g(Q)\ ] = O(\text{poly}(n)e^{-n^\epsilon})$ for any query Q. By observation 5.1.1, this suffices to prove the theorem.

Factor n approximation algorithm for MS-MST: We are given a graph $G(V, E)$ and submodular cost functions f_i for each agent. Define a new weight function w over E as $w_e = \min_i f_i(\{e\})$. Run Kruskal's algorithm on G with weight function $w(Z) = \sum_{e \in Z} w_e$ for all $Z \subseteq E$ to get a minimum spanning tree T. Assign each edge $e \in T$ to the agent i minimizing $f_i(\{e\})$. The proof that this constitutes an n-approximate solution follows similar arguments as in the proof of Claim 38. □

5.6 Discussion

The setting that we have considered in this work is quite general, and is a very exciting avenue of research. There are many other interesting problems in this class like shortest path, minimum cut, minimum edge cover etc which could be studied in the future work. We have considered the covering problems in this contribution, one can ask the same questions for packing problems like maximum matching.

One could also consider even more general functions like Subadditive functions. Extension to multi-agent makes a natural connection to Game Theory. Algorithmic mechanism design of these combinatorial problem has interesting applications.

Another area of interest is the characterization of problems for which the computational complexity doesn't change when considered in this general setting compared to the linear one.

REFERENCES

[1] Andelman, N., *Online and strategic aspects of network resource management algorithms*. PhD thesis, Tel Aviv University, Tel Aviv, Israel, 2006.

[2] Andelman, N. and Mansour, Y., "Auctions with budget constraints," in *9th Scandinavian Workshop on Algorithm Theory (SWAT)*, pp. 26–38, 2004.

[3] Azar, Y., Birnbaum, B. E., Karlin, A. R., Mathieu, C., and Nguyen, C. T., "Improved approximation algorithms for budgeted allocations," in *ICALP*, pp. 186–197, 2008.

[4] Babaioff, M., Immorlica, N., and Kleinberg, R., "Matroids, secretary problems, and online mechanisms," in *SODA*, 2007.

[5] Bar-Yehuda, R. and Even, S., "A linear time approximation algorithm for the weighted vertex cover problem," *Journal of Algorithms*, vol. 2, pp. 198–203, 1981.

[6] Benoit, J.-P. and Krishna, V., "Multiple object auctions with budget constrained bidders," *Review of Economic Studies*, vol. 68, pp. 155–179, 2001.

[7] Blumrosen, L. and Nisan, N., "Combinatorial auctions," in *Algorithmic Game Theory* (Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V., eds.), pp. 267–299, Cambridge University Press, 2007.

[8] Bollobas, B., *Random Graphs*. Cambridge University Press, 2001.

[9] Buchbinder, N., Jain, K., and Naor, S., "Online primal-dual algorithms for maximizing ad-auctions revenue.," *Proceedings of ESA*, 2007.

[10] Calinescu, G., Chekuri, C., Pál, M., and Vondrák, J., "Maximizing a submodular set function subject to a matroid constraint (extended abstract)," in *IPCO*, pp. 182–196, 2007.

[11] Carr, R. and Vempala, S., "Randomized metarounding," *Random Struct. and Algorithms*, vol. 20, pp. 343–352, 2002.

[12] Chakrabarty, D., *Algorithmic Aspects of Connectivity, Allocation and Design Problems*. PhD thesis, Georgia Institute of Technology, 2008.

[13] Chakrabarty, D. and Goel, G., "On the approximability of budgeted allocations and improved lower bounds for submodular welfare maximization and gap," in *FOCS*, pp. 687–696, 2008.

[14] Chekuri, C. and Khanna, S., "A PTAS for the multiple-knapsack problem," *Proceedings of SODA*, 2000.

[15] Chen, N., Engelberg, R., Nguyen, C., Raghavendra, P., Rudra, A., and Singh, G., "Improved approximation algorithms for the spanning star forest problem," *Proceedings of APPROX*, 2007.

[16] Chlebik, M. and Chlebikova, J., "Approximation hardness for small occurrence instances of np-hard problems.," *Proceedings of CIAC*, 2003.

[17] Devanur, N. and Vazirani, V., "The spending constraint model for market equilibrium: algorithmic, existence and uniqueness results," in *STOC*, pp. 519–528, 2004.

[18] Dinur, I. and Safra, S., "On the hardness of approximating minimum vertex cover," *Annals of Mathematics*, vol. 162, p. 2005, 2005.

[19] Dynkin, E. B., "The optimum choice of the instant for stopping a markov process," *Sov. Math. Dokl.*, vol. 4, 1963.

[20] Feige, U. and Vondrák, J., "Approximation algorithms for allocation problems: Improving the factor of 1 - 1/e," *Proceedings of FOCS*, 2006.

[21] Feige, U., "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, 1998.

[22] Feige, U., Mirrokni, V. S., and Vondrák, J., "Maximizing non-monotone submodular functions," in *FOCS*, pp. 461–471, 2007.

[23] Fleischer, L., Goemans, M. X., Mirrokini, V., and Sviridenko, M., "Tight approximation algorithms for maximum general assignment problems,," *Proceedings of SODA*, 2006.

[24] Garg, R., Kumar, V., and Pandit, V., "Approximation algorithms for budget-constrained auctions," *Proceedings of APPROX*, 2001.

[25] Goel, G. and Mehta, A., "Adwords auctions with decreasing valuation bids," in *WINE*, pp. 335–340, 2007.

[26] Goel, G. and Mehta, A., "Online budgeted matching in random input models with applications to adwords," in *SODA*, pp. 982–991, 2008.

[27] Goemans, M. X., Harvey, N. J. A., Iwata, S., and Mirrokni, V. S., "Approximating submodular functions everywhere," in *SODA*, pp. 535–544, 2009.

[28] Guha, S. and McGregor, A., "Approximate quantiles and the order of the stream," in *PODS*, pp. 273–279, 2006.

[29] Hrastad, J., "Some optimal inapproximability results," *JACM*, vol. 48, pp. 798–859, 2001.

[30] Hayrapetyan, A., Swamy, C., and Tardos, É., "Network design for information networks," in *SODA*, pp. 933–942, 2005.

[31] Iwata, S., "A faster scaling algorithm for minimizing submodular functions," *SIAM J. Comput.*, vol. 32, no. 4, pp. 833–840, 2003.

[32] Iwata, S., "Submodular function minimization," *Math. Program.*, vol. 112, no. 1, pp. 45–64, 2008.

[33] Iwata, S., Fleischer, L., and Fujishige, S., "A combinatorial strongly polynomial algorithm for minimizing submodular functions," *J. ACM*, vol. 48, no. 4, pp. 761–777, 2001.

[34] Iwata, S. and Orlin, J. B., "A simple combinatorial algorithm for submodular function minimization," in *SODA '09: Proceedings of the Nineteenth Annual ACM -SIAM Symposium on Discrete Algorithms*, pp. 1230–1237, 2009.

[35] Jain, K., "Factor 2 approximation algorithm for the generalized steiner network problem," *Proceedings of FOCS*, 1998.

[36] Kalyanasundaram, B. and Pruhs, K. R., "An optimal deterministic algorithm for online b -matching," *Theoretical Computer Science*, vol. 233, no. 1–2, pp. 319–325, 2000.

[37] Karp, R., Vazirani, U., and Vazirani, V., "An optimal algorithm for online bipartite matching," in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, 1990.

[38] Khot, S., Lipton, R., Markakis, E., and Mehta, A., "Inapproximability results for combinatorial auctions with submodular utility functions," *Proceedings of WINE*, 2005.

[39] Khot, S., "On the power of unique 2-prover 1-round games," in *STOC*, pp. 767–775, 2002.

[40] Khot, S., Lipton, R. J., Markakis, E., and Mehta, A., "Inapproximability results for combinatorial auctions with submodular utility functions," *Algorithmica*, vol. 52, no. 1, pp. 3–18, 2008.

[41] Khot, S. and Regev, O., "Vertex cover might be hard to approximate to within 2-epsilon," *J. Comput. Syst. Sci.*, vol. 74, no. 3, pp. 335–349, 2008.

[42] Krohn, E. and Varadarajan, K. *Private Communication*, 2007.

[43] Lahaie, S., Pennock, D., Saberi, A., and Vohra, R., *Algorithmic Game Theory (Nisan et al. editors)*, ch. Sponsored Search. 2007.

[44] Lau, L. C., Naor, S., Salavatipour, M., and Singh, M., "Survivable network design with degree or order constraints.," *Proceedings of STOC*, 2007.

[45] Lehman, B., Lehman, D., and Nisan, N., "Combinatorial auctions with decreasing marginal utilities," in *Proceedings of the 3rd ACM conference on Electronic Commerce*, pp. 18 –28, 2001.

[46] Lenstra, J. K., Shmoys, D., and Tardos, E., "Approximation algorithms for scheduling unrelated parallel machines," *Math. Programming*, vol. 46, pp. 259–271, 1990.

[47] Mahdian, M., Nazerzadeh, H., and Saberi, A., "Allocating online advertisement space with unreliable estimates," in *ACM Conference on Electronic Commerce*, 2007.

[48] Mehta, A., Saberi, A., Vazirani, U., and Vazirani, V., "Adwords and generalized online matching," in *FOCS*, 2005.

[49] Mirrokni, V. S., Schapira, M., and Vondrák, J., "Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions," in *ACM Conference on Electronic Commerce*, pp. 70–77, 2008.

[50] Nguyen, C., Shen, J., Hou, M. M., Sheng, L., Miller, W., and Zhang, L., "Approximating the spanning star forest problem and its applications to genomic sequence alignment," *Proceedings of SODA*, 2007.

[51] Nisan, N., *Combinatorial Auctions by Cramton, Shoham and Steinberg*, ch. Bidding languages for combinatorial auctions. 2005.

[52] Nisan, N., "The communication complexity of approximate set packing and covering," in *ICALP*, pp. 868–875, 2002.

[53] Orlin, J. B., "A faster strongly polynomial time algorithm for submodular function minimization," in *IPCO*, pp. 240–251, 2007.

[54] Sandholm, T. and Suri, S., "Side constraints and non-price attributes in markets," *Proceedings of IJCAI*, 2001.

[55] Schrijver, A., "A combinatorial algorithm minimizing submodular functions in strongly polynomial time," *J. Comb. Theory, Ser. B*, vol. 80, no. 2, pp. 346–355, 2000.

[56] Sharma, Y., Swamy, C., and Williamson, D. P., "Approximation algorithms for prize collecting forest problems with submodular penalty functions," in *SODA*, pp. 1275–1284, 2007.

[57] Shmoys, D. and Tardos, E., "An approximation algorithm for the generalized assignment problem," *Math. Programming*, vol. 62, pp. 461–474, 1993.

[58] Singh, M. and Lau, L. C., "Approximating minimum bounded degree spanning trees to within one of optimal.," *Proceedings of STOC*, 2007.

[59] Srinivasan, A., "Budgeted allocations in the full-information setting," in *APPROX-RANDOM*, pp. 247–253, 2008.

[60] Sviridenko, M., "A note on maximizing a submodular set function subject to a knapsack constraint," *Oper. Res. Lett.*, vol. 32, no. 1, pp. 41–43, 2004.

[61] Svitkina, Z. and Fleischer, L., "Submodular approximation: Sampling-based algorithms and lower bounds," in *FOCS*, pp. 697–706, 2008.

[62] Svitkina, Z. and Tardos, É., "Facility location with hierarchical facility costs," in *SODA*, pp. 153–161, 2006.

[63] Vazirani, V. V., **Approximation Algorithms**. Springer, 2002.

[64] Vondrák, J., "Optimal approximation for the submodular welfare problem in the value oracle model," *Proceedings of STOC*, 2008.

[65] Wolsey, L. A., "An analysis of the greedy algorithm for the submodular set covering problem," in *Combinatorica*, pp. 385–393, 1982.

[66] Yao, A. C., "Probabilistic computations: towards a unified measure of complexity," *FOCS*, pp. 222–227, 1977.